Cygwin User's Guide

Cygwin User’s Guide

Table of Contents

List of Examples

Chapter 1. Cygwin Overview

1.1. What is it?

Cygwin is a Linux-like environment for Windows. It consists of a DLL (`cygwin1.dll`), which acts as an emulation layer providing substantial POSIX (http://www.pasc.org/#POSIX) (Portable Operating System Interface) system call functionality, and a collection of tools, which provide a Linux look and feel. The Cygwin DLL works with all x86 and AMD64 versions of Windows NT since Windows NT 4. The API follows the Single Unix Specification (http://www.opengroup.org/onlinepubs/009695399/nfindex.html) as much as possible, and then Linux practice. The major differences between Cygwin and Linux is the C library (`newlib` instead of `glibc`).

With Cygwin installed, users have access to many standard UNIX utilities. They can be used from one of the provided shells such as **bash** or from the Windows Command Prompt. Additionally, programmers may write Win32 console or GUI applications that make use of the standard Microsoft Win32 API and/or the Cygwin API. As a result, it is possible to easily port many significant UNIX programs without the need for extensive changes to the source code. This includes configuring and building most of the available GNU software (including the development tools included with the Cygwin distribution).

1.2. Quick Start Guide for those more experienced with Windows

If you are new to the world of UNIX, you may find it difficult to understand at first. This guide is not meant to be comprehensive, so we recommend that you use the many available Internet resources to become acquainted with UNIX basics (search for "UNIX basics" or "UNIX tutorial").

To install a basic Cygwin environment, run the **setup.exe** program and click `Next` at each page. The default settings are correct for most users. If you want to know more about what each option means, see Section 2.1. Use **setup.exe** any time you want to update or install a Cygwin package. If you are installing Cygwin for a specific purpose, use it to install the tools that you need. For example, if you want to compile C++ programs, you need the gcc-g++ package and probably a text editor like nano. When running **setup.exe**, clicking on categories and packages in the package installation screen will provide you with the ability to control what is installed or updated.

Another option is to install everything by clicking on the `Default` field next to the `All` category. However, be advised that this will download and install several hundreds of megabytes of software to your computer. The best plan is probably to click on individual categories and install either entire categories or packages from the categories themselves. After installation, you can find Cygwin-specific documentation in the `/usr/share/doc/Cygwin/` directory.

Developers coming from a Windows background will be able to write console or GUI executables that rely on the Microsoft Win32 API instead of Cygwin using the -mno-cygwin option to GCC. The **-shared** option allows to write Windows Dynamically Linked Libraries (DLLs). The resource compiler **windres** is also provided.

1.3. Quick Start Guide for those more experienced with UNIX

If you are an experienced UNIX user who misses a powerful command-line environment, you will enjoy Cygwin. Developers coming from a UNIX background will find a set of utilities they are already comfortable using, including a working UNIX shell. The compiler tools are the standard GNU compilers most people will have previously used under UNIX, only ported to the Windows host. Programmers wishing to port UNIX software to Windows NT will find that the Cygwin library provides an easy way to port many UNIX packages, with only minimal source code changes.

Note that there are some workarounds that cause Cygwin to behave differently than most UNIX-like operating systems; these are described in more detail in Section 3.8.

Use the graphical command **setup.exe** any time you want to update or install a Cygwin package. This program must be run manually every time you want to check for updated packages since Cygwin does not currently include a mechanism for automatically detecting package updates.

By default, **setup.exe** only installs a minimal subset of packages. Add any other packages by clicking on the + next to the Category name and selecting the package from the displayed list. You may search for specfic tools by using the Setup Package Search (http://cygwin.com/packages/) at the Cygwin web site.

Another option is to install everything by clicking on the `Default` field next to the `All` category. However, be advised that this will download and install several hundreds of megabytes of software to your computer. The best plan is probably to click on individual categories and install either entire categories or packages from the categories themselves. After installation, you can find Cygwin-specific documentation in the `/usr/share/doc/Cygwin/` directory.

For more information about what each option in **setup.exe** means, see Section 2.1.

1.4. Are the Cygwin tools free software?

Yes. Parts are GNU (http://www.gnu.org/) software (**gcc**, **gas**, **ld**, etc.), parts are covered by the standard X11 license (http://www.x.org/Downloads_terms.html), some of it is public domain, some of it was written by Red Hat and placed under the GNU General Public License (http://www.gnu.org/licenses/gpl.html) (GPL). None of it is shareware. You don't have to pay anyone to use it but you should be sure to read the copyright section of the FAQ for more information on how the GNU GPL may affect your use of these tools. If you intend to port a proprietary application using the Cygwin library, you may want the Cygwin proprietary-use license. For more information about the proprietary-use license, please go to http://www.redhat.com/software/tools/cygwin/. Customers of the native Win32 GNUPro should feel free to submit bug reports and ask questions through the normal channels. All other questions should be sent to the project mailing list <`cygwin@cygwin.com`>.

1.5. A brief history of the Cygwin project

> **Note:** A historical look into the first years of Cygwin development is Geoffrey J. Noer's 1998 paper, "Cygwin32: A Free Win32 Porting Layer for UNIX® Applications" which can be found at the 2nd USENIX Windows NT Symposium Online Proceedings (http://www.usenix.org/publications/library/proceedings/usenix-nt98/technical.html).

Cygwin began development in 1995 at Cygnus Solutions (now part of Red Hat, Inc.). The first thing done was to enhance the development tools (**gcc**, **gdb**, **gas**, etc.) so that they could generate and interpret Win32 native object files. The next task was to port the tools to Win NT/9x. We could have done this by rewriting large portions of the source to work within the context of the Win32 API. But this would have meant spending a huge amount of time on each and every tool. Instead, we took a substantially different approach by writing a shared library (the Cygwin DLL) that adds the necessary UNIX-like functionality missing from the Win32 API (`fork`, `spawn`, `signals`, `select`, `sockets`, etc.). We call this new interface the Cygwin API. Once written, it was possible to build working Win32 tools using UNIX-hosted cross-compilers, linking against this library.

From this point, we pursued the goal of producing native tools capable of rebuilding themselves under Windows 9x and NT (this is often called self-hosting). Since neither OS ships with standard UNIX user tools (fileutils, textutils, bash, etc...), we had to get the GNU equivalents working with the Cygwin API. Most of these tools were previously only built natively so we had to modify their configure scripts to be compatible with cross-compilation. Other than the configuration changes, very few source-level changes had to be made. Running bash with the development tools and user tools in place, Windows 9x and NT look like a flavor of UNIX from the perspective of the GNU configure mechanism. Self hosting was achieved as of the beta 17.1 release in October 1996.

The entire Cygwin toolset was available as a monolithic install. In April 2000, the project announced a New Cygwin Net Release (http://www.cygwin.com/ml/cygwin/2000-04/msg00269.html) which provided the native Win32 program **setup.exe** to install and upgrade each package separately. Since then, the Cygwin DLL and **setup.exe** have seen continuous development.

The latest major improvement in this development is the 1.7 release in 2009, which dropped Windows 95/98/Me support in favor of using Windows NT features more extensively. It adds a lot of new features like case-sensitive filenames, NFS interoperability, IPv6 support and much more.

1.6. Highlights of Cygwin Functionality

1.6.1. Introduction

When a binary linked against the library is executed, the Cygwin DLL is loaded into the application's text segment. Because we are trying to emulate a UNIX kernel which needs access to all processes running under it, the first Cygwin DLL to run creates shared memory areas and global synchronization objects that other processes using separate instances of the DLL can access. This is used to keep track of open file descriptors and to assist fork and exec, among other purposes. Every process also has a per_process structure that contains information such as process id, user id, signal masks, and other similar process-specific information.

The DLL is implemented as a standard DLL in the Win32 subsystem. Under the hood it's using the Win32 API, as well as the native NT API, where appropriate.

Because processes run under the standard Win32 subsystem, they can access both the UNIX compatibility calls provided by Cygwin as well as any of the Win32 API calls. This gives the programmer complete flexibility in designing the structure of their program in terms of the APIs used. For example, they could write a Win32-specific GUI using Win32 API calls on top of a UNIX back-end that uses Cygwin.

The native NT API is used mainly for speed, as well as to access NT capabilities which are useful to implement certain POSIX features, but are hidden to the Win32 API.

Due to some restrictions in Windows, it's not always possible to strictly adhere to existing UNIX standards like POSIX.1. Fortunately these are mostely border cases.

1.6.2. Permissions and Security

Windows NT includes a sophisticated security model based on Access Control Lists (ACLs). Cygwin maps Win32 file ownership and permissions to ACLs by default, on file systems supporting them (usually NTFS). Solaris style ACLs and accompanying function calls are also supported. The chmod call maps UNIX-style permissions back to the Win32 equivalents. Because many programs expect to be able to find the `/etc/passwd` and `/etc/group` files, we provide utilities (http://cygwin.com/cygwin-ug-net/using-utils.html) that can be used to construct them from the user and group information provided by the operating system.

Users with Administrator rights are permitted to chown files. With version 1.1.3 Cygwin introduced a mechanism for setting real and effective UIDs. This is described in Section 2.5. As of version 1.5.13, the Cygwin developers are not aware of any feature in the Cygwin DLL that would allow users to gain privileges or to access objects to which they have no rights under Windows. However there is no guarantee that Cygwin is as secure as the Windows it runs on. Cygwin processes share some variables and are thus easier targets of denial of service type of attacks.

1.6.3. File Access

Cygwin supports both POSIX- and Win32-style paths, using either forward or back slashes as the directory delimiter. Paths coming into the DLL are translated from POSIX to native NT as needed. From the application perspective, the file system is a POSIX-compliant one. The implementation details are safely hidden in the Cygwin DLL. UNC pathnames (starting with two slashes) are supported for network paths.

Since version 1.7.0, the layout of this POSIX view of the Windows file system space is stored in the `/etc/fstab` file. Actually, there is a system-wide `/etc/fstab` file as well as a user-specific fstab file `/etc/fstab.d/${USER}`.

At startup the DLL has to find out where it can find the `/etc/fstab` file. The mechanism used for this is simple. First it retrieves it's own path, for instance `C:\Cygwin\bin\cygwin1.dll`. From there it deduces that the root path is `C:\Cygwin`. So it looks for the `fstab` file in `C:\Cygwin\etc\fstab`. The layout of this file is very similar to the layout of the `fstab` file on Linux. Just instead of block devices, the mount points point to Win32 paths. An installation with **setup.exe** installs a `fstab` file by default, which can easily be changed using the editor of your choice.

In addition to selecting the root partition, the `fstab` file allows mounting arbitrary Win32 paths into the POSIX file system space. A special case is the so-called cygdrive prefix. It's the path under which every available drive in the system is mounted under its drive letter. The default value is `/cygdrive`, so you can access the drives as `/cygdrive/c`, `/cygdrive/d`, etc... The cygdrive prefix can be set to some other value (`/mnt` for instance) in the `fstab` file(s).

The library exports several Cygwin-specific functions that can be used by external programs to convert a path or path list from Win32 to POSIX or vice versa. Shell scripts and Makefiles cannot call these functions directly. Instead, they can do the same path translations by executing the **cygpath** utility program that we provide with Cygwin.

Win32 applications handle filenames in a case preserving, but case insensitive manner. Cygwin supports case sensitivity on file systems supporting that. Since Windows XP, the OS only supports case sensitivity when a specific registry value is changed. Therefore, case sensitivity is not usually the default.

Symbolic links are not present and supported on Windows up to and including Windows Server 2003 R2. Native symlinks are available starting with Windows Vista. Due to their strange implementation, however, they are not useful in a POSIX emulation layer. Cygwin recognizes native symlinks, but does not create them.

Symbolic links are potentially created in two different ways. The file style symlinks are files containing a magic cookie followed by the path to which the link points. They are marked with the System DOS attribute so that only files with that attribute have to be read to determine whether or not the file is a symbolic link. The shortcut style symlinks are Windows shortcut files with a special header and the Readonly DOS attribute set. The advantage of file symlinks is speed, the advantage of shortcut symlinks is the fact that they can be utilized by non-Cygwin Win32 tools as well.

Starting with Cygwin 1.7, symbolic links are using UTF-16 to encode the filename of the target file, to better support internationalization. Symlinks created by older Cygwin releases can be read just fine. However, you could run into problems with them if you're now using another character set than the one you used when creating these symlinks (see Section 2.4.4. Please note that this new UTF-16 style of symlinks is not compatible with older Cygwin release, which can't read the target filename correctly.

Hard links are fully supported on NTFS and NFS file systems. On FAT and other file systems which don't support hardlinks, the call returns with an error, just like on other POSIX systems.

On file systems which don't support unique persistent file IDs (FAT, older Samba shares) the inode number for a file is calculated by hashing its full Win32 path. The inode number generated by the stat call always matches the one returned in `d_ino` of the `dirent` structure. It is worth noting that the number produced by this method is not guaranteed to be unique. However, we have not found this to be a significant problem because of the low probability of generating a duplicate inode number.

`chroot(2)` is supported since Cygwin 1.1.3. However, chroot is not a concept known by Windows. This implies some restrictions. First of all, the `chroot` call isn't a privileged call. Any user may call it. Second, the chroot environment isn't safe against native windows processes. If you want to use a chroot environment to, for example, allow anonymous ftp with restricted access, you must make sure care that only native Cygwin applications are accessible inside of the chroot environment. Since those applications are only using the Cygwin POSIX API to access the file system their access can be restricted as it is intended. This includes not only POSIX paths but Win32 paths containing drive letter and/or backslashes as well as UNC paths (`//server/share` or `\\server\share`).

1.6.4. Text Mode vs. Binary Mode

It is often important that files created by native Windows applications be interoperable with Cygwin applications. For example, a file created by a native Windows text editor should be readable by a Cygwin application, and vice versa.

Unfortunately, UNIX and Win32 have different end-of-line conventions in text files. A UNIX text file will have a single newline character (LF) whereas a Win32 text file will instead use a two character sequence (CR+LF). Consequently, the two character sequence must be translated on the fly by Cygwin into a single character newline when reading in text mode.

This solution addresses the newline interoperability concern at the expense of violating the POSIX requirement that text and binary mode be identical. Consequently, processes that attempt to lseek through text files can no longer rely on the number of bytes read to be an accurate indicator of position within the file. For this reason, Cygwin allows you to choose the mode in which a file is read in several ways.

1.6.5. ANSI C Library

We chose to include Red Hat's own existing ANSI C library "newlib" as part of the library, rather than write all of the lib C and math calls from scratch. Newlib is a BSD-derived ANSI C library, previously only used by cross-compilers for embedded systems development. Other functions, which are not supported by newlib have been added to the Cygwin sources using BSD implementations as much as possible.

The reuse of existing free implementations of such things as the glob, regexp, and getopt libraries saved us considerable effort. In addition, Cygwin uses Doug Lea's free malloc implementation that successfully balances speed and compactness. The library accesses the malloc calls via an exported function pointer. This makes it possible for a Cygwin process to provide its own malloc if it so desires.

1.6.6. Process Creation

The `fork` call in Cygwin is particularly interesting because it does not map well on top of the Win32 API. This makes it very difficult to implement correctly. Currently, the Cygwin fork is a non-copy-on-write implementation similar to what was present in early flavors of UNIX.

The first thing that happens when a parent process forks a child process is that the parent initializes a space in the Cygwin process table for the child. It then creates a suspended child process using the Win32 CreateProcess call. Next, the parent process calls setjmp to save its own context and sets a pointer to this in a Cygwin shared memory area (shared among all Cygwin tasks). It then fills in the child's .data and .bss sections by copying from its own address space into the suspended child's address space. After the child's address space is initialized, the child is run while the parent waits on a mutex. The child discovers it has been forked and longjumps using the saved jump buffer. The child then sets the mutex the parent is waiting on and blocks on another mutex. This is the signal for the parent to copy its stack and heap into the child, after which it releases the mutex the child is waiting on and returns from the fork call. Finally, the child wakes from blocking on the last mutex, recreates any memory-mapped areas passed to it via the shared area, and returns from fork itself.

While we have some ideas as to how to speed up our fork implementation by reducing the number of context switches between the parent and child process, fork will almost certainly always be inefficient under Win32. Fortunately, in most circumstances the spawn family of calls provided by Cygwin can be substituted for a fork/exec pair with only a little effort. These calls map cleanly on top of the Win32 API. As a result, they are much more efficient. Changing the compiler's driver program to call spawn instead of fork was a trivial change and increased compilation speeds by twenty to thirty percent in our tests.

However, spawn and exec present their own set of difficulties. Because there is no way to do an actual exec under Win32, Cygwin has to invent its own Process IDs (PIDs). As a result, when a process performs multiple exec calls, there will be multiple Windows PIDs associated with a single Cygwin PID. In some cases, stubs of each of these Win32 processes may linger, waiting for their exec'd Cygwin process to exit.

1.6.7. Signals

When a Cygwin process starts, the library starts a secondary thread for use in signal handling. This thread waits for Windows events used to pass signals to the process. When a process notices it has a signal, it scans its signal bitmask and handles the signal in the appropriate fashion.

Several complications in the implementation arise from the fact that the signal handler operates in the same address space as the executing program. The immediate consequence is that Cygwin system functions are interruptible unless special care is taken to avoid this. We go to some lengths to prevent the sig_send function that sends signals from being interrupted. In the case of a process sending a signal to another process, we place a mutex around sig_send such that sig_send will not be interrupted until it has completely finished sending the signal.

In the case of a process sending itself a signal, we use a separate semaphore/event pair instead of the mutex. sig_send starts by resetting the event and incrementing the semaphore that flags the signal handler to process the signal. After the signal is processed, the signal handler signals the event that it is done. This process keeps intraprocess signals synchronous, as required by POSIX.

Most standard UNIX signals are provided. Job control works as expected in shells that support it.

1.6.8. Sockets

Socket-related calls in Cygwin basically call the functions by the same name in Winsock, Microsoft's implementation of Berkeley sockets, but with lots of tweaks. All sockets are non-blocking under the hood to allow to interrupt blocking calls by POSIX signals. Additional bookkeeping is necessary to implement correct socket sharing POSIX semantics and especially for the select call. Some socket-related functions are not implemented at all in Winsock, as, for example, socketpair. Starting with Windows Vista, Microsoft removed the legacy calls `rcmd(3)`, `rexec(3)` and `rresvport(3)`. Recent versions of Cygwin now implement all these calls internally.

An especially troublesome feature of Winsock is that it must be initialized before the first socket function is called. As a result, Cygwin has to perform this initialization on the fly, as soon as the first socket-related function is called by the application. In order to support sockets across fork calls, child processes initialize Winsock if any inherited file descriptor is a socket.

AF_UNIX (AF_LOCAL) sockets are not available in Winsock. They are implemented in Cygwin by using local AF_INET sockets instead. This is completely transparent to the application. Cygwin's implementation also supports the getpeereid BSD extension. However, Cygwin does not yet support descriptor passing.

IPv6 is supported beginning with Cygwin release 1.7.0. This support is dependent, however, on the availability of the Windows IPv6 stack. The IPv6 stack was "experimental", i.e. not feature complete in Windows 2003 and earlier. Full IPv6 support became available starting with Windows Vista and Windows Server 2008. Cygwin does not depend on the underlying OS for the (newly implemented) `getaddrinfo` and `getnameinfo` functions. Cygwin 1.7.0 adds replacement functions which implement the full functionality for IPv4.

1.6.9. Select

The UNIX `select` function is another call that does not map cleanly on top of the Win32 API. Much to our dismay, we discovered that the Win32 select in Winsock only worked on socket handles. Our implementation allows select to function normally when given different types of file descriptors (sockets, pipes, handles, and a custom /dev/windows Windows messages pseudo-device).

Upon entry into the select function, the first operation is to sort the file descriptors into the different types. There are then two cases to consider. The simple case is when at least one file descriptor is a type that is always known to be ready (such as a disk file). In that case, select returns immediately as soon as it has polled each of the other types to see if they are ready. The more complex case involves waiting for socket or pipe file descriptors to be ready. This is accomplished by the main thread suspending itself, after starting one thread for each type of file descriptor present. Each thread polls the file descriptors of its respective type with the appropriate Win32 API call. As soon as a thread identifies a ready descriptor, that thread signals the main thread to wake up. This case is now the same as the first one since we know at least one descriptor is ready. So select returns, after polling all of the file descriptors one last time.

1.7. What's new and what changed in Cygwin 1.7

1.7.1. OS related changes

```
- Windows 95, 98 and Me are not supported anymore.  The new Cygwin 1.7 DLL
  will not run on any of these systems.
```

1.7.2. File Access related changes

```
- Mount points are no longer stored in the registry.  Use /etc/fstab
  and /etc/fstab.d/$USER instead.  Mount points created with mount(1)
  are only local to the current session and disappear when the last
```

```
  Cygwin process in the session exits.

- Cygwin creates the mount points for /, /usr/bin, and /usr/lib automatically
  from it's own position on the disk.  They don't have to be specified in
  /etc/fstab.

- If a filename cannot be represented in the current character set,
  the character will be converted to a sequence Ctrl-X + UTF-8 representation
  of the character.  This allows to access all files, even those not
  having a valid representation of their filename in the current character
  set (codepage).  To always have a valid string, use the UTF-8 charset
  by setting the environment variable $LANG, $LC_ALL, or $LC_CTYPE to a
  valid POSIX value, for instance in Cygwin.bat like this:

    set LC_CTYPE=en_US.UTF-8

- PATH_MAX is now 4096.  Internally, path names can be as long as the
  underlying OS can handle (32K).

- struct dirent now supports d_type, filled out with DT_REG or DT_DIR.
  All other file types return as DT_UNKNOWN for performance reasons.

- The CYGWIN environment variable options "ntsec" and "smbntsec" have
  been replaced by the per-mount option "acl"/"noacl".

- The CYGWIN environment variable option "ntea" has been removed without
  substitute.

- The CYGWIN environment variable option "check_case" has been removed
  in favor of real case-sensitivity on file systems supporting it.

- Creating filenames with special DOS characters '"', '*', ':', '<',
  '>', '|' is supported.

- Creating files with special DOS device filename components ("aux",
  "nul", "prn") is supported.

- File names are case sensitive if the OS and the underlying file system
  supports it.  Works on NTFS and NFS.  Does not work on FAT and Samba
  shares.  Requires to change a registry key (see the user's guide).
  Can be switched off on a per-mount base.

- Due to the above changes, managed mounts have been removed.

- Incoming DOS paths are always handled case-insensitive and get no POSIX
  permission, as if they are mounted with noacl,posix=0 mount flags.

- unlink(2) and rmdir(2) try very hard to remove files/directories even
  if they are currently accessed or locked.  This is done by utilizing
  the hidden recycle bin directories and marking the files for deletion.

- rename(2) rewritten to be more POSIX conformant.
```

```
- access(2) now performs checks using the real user ID, as required by
  POSIX; the old behavior of querying based on effective user ID is
  available through the new faccessat(2) and euidaccess(2) APIs.

- Add st_birthtim member to struct stat.

- File locking is now advisory, not mandatory anymore.  The fcntl(2) and
  the new lockf(2) APIs create and maintain locks with POSIX semantics,
  the flock(2) API creates and maintains locks with BSD semantics.
  POSIX and BSD locks are independent of each other.

- Implement atomic O_APPEND mode.

- New open(2) flags O_DIRECTORY, O_EXEC and O_SEARCH.

- Make the "plain file with SYSTEM attribute set" style symlink
  default again when creating symlinks.  Only create Windows shortcut
  style symlinks if CYGWIN=winsymlinks is set in the environment.

- Symlinks now use UTF-16 encoding for the target filename for
  better internationalization support.  Cygwin 1.7 can read all old style
  symlinks, but the new style is not compatible with older Cygwin releases.

- Handle NTFS native symlinks available since Vista/2008 as symlinks
  (but don't create Vista/2008 symlinks due to unfortunate OS restrictions).

- Recognize NFS shares and handle them using native mechanisms.
  Recognize and create real symlinks on NFS shares.  Get correct
  stat(2) information and set real mode bits on open(2), mkdir(2)
  and chmod(2).

- Recognize Netapp DataOnTap drives and fix inode number handling.

- Recognize Samba version beginning with Samba 3.0.28a using the new
  extended version information negotiated with the Samba developers.

- Stop faking hardlinks by copying the file on filesystems which don't
  support hardlinks natively (FAT, FAT32, etc.).  Just return an error
  instead, just like Linux.

- List servers of all accessible domains and workgroups in // instead of
  just the servers in the own domain/workgroup.

- Support Linux-like extended attributes ([fl]getxattr, [fl]listxattr,
  [fl]setxattr, [fl]removexattr).

- New file conversion API for conversion from Win32 to POSIX path and
  vice versa (cygwin_conv_path, cygwin_create_path, cygwin_conv_path_list).

- New openat family of functions: openat, faccessat, fchmodat, fchownat,
  fstatat, futimesat, linkat, mkdirat, mkfifoat, mknodat, readlinkat, renameat,
  symlinkat, unlinkat.
```

- Other new APIs: posix_fadvise, posix_fallocate, funopen, fopencookie, open_memstream, open_wmemstream, fmemopen, fdopendir, fpurge, mkstemps, eaccess, euidaccess, canonicalize_file_name, fexecve, execvpe.

1.7.3. Network related changes

- New implementation for blocking sockets and select on sockets which is supposed to allow POSIX-compatible sharing of sockets between threads and processes.

- send/sendto/sendmsg now send data in 64K chunks to circumvent an internal buffer problem in WinSock (KB 201213).

- New send/recv option MSG_DONTWAIT.

- IPv6 support. New API getaddrinfo, getnameinfo, freeaddrinfo, gai_strerror, in6addr_any, in6addr_loopback. On IPv6-less systems, replacement functions are available for IPv4. On systems with IPv6 enabled, the underlying WinSock functions are used. While I tried hard to get the functionality as POSIXy as possible, keep in mind that a *fully* conformant implementation of getaddrinfo and other stuff is only available starting with Windows Vista/2008.

- Resolver functions (res_init, res_query, res_search, res_querydomain, res_mkquery, res_send, dn_comp, dn_expand) are now part of Cygwin. Applications don't have to link against minires anymore. Actually, this *is* the former libminires.a.

- rcmd is now implemented inside of Cygwin, instead of calling the WinSock function. This allows rsh(1) usage on Vista/2008 and later, which dropped this function from WinSock.

- Define multicast structures in netinet/in.h. Note that fully conformant multicast support is only available beginning with Vista/2008.

- Improve get_ifconf. Redefine struct ifreq and subsequent datastructures to be able to keep more information. Support SIOCGIFINDEX, SIOCGIFDSTADDR and the Cygwin specific SIOCGIFFRNDLYNAM. Support real interface flags on systems supporting them.

- Other new APIs: bindresvport, bindresvport_sa, gethostbyname2, iruserok_sa, rcmd_af, rresvport_af. getifaddrs, freeifaddrs, if_nametoindex, if_indextoname, if_nameindex, if_freenameindex.

- Add /proc/net/if_inet6.

1.7.4. Device related changes

```
- Reworked pipe implementation which uses overlapped IO to create
  more reliable interruptible pipes and fifos.

- The CYGWIN environment variable option "binmode" has been removed.

- Improved fifo handling by using native Windows named pipes.

- Detect when a stdin/stdout which looks like a pipe is really a tty.
  Among other things, this allows a debugged application to recognize that
  it is using the same tty as the debugger.

- Support UTF-8 in console window.

- In the console window the backspace key now emits DEL (0x7f) instead
  of BS (0x08), Alt-Backspace emits ESC-DEL (0x1b,0x7f) instead of DEL
  (0x7f), same as the Linux console and xterm.
  Control-Space now emits an ASCII NUL (0x0) character.

- Support up to 64 serial interfaces using /dev/ttyS0 - /dev/ttyS63.

- Support up to 128 raw disk drives /dev/sda - /dev/sddx.

- New API: cfmakeraw, posix_openpt.
```

1.7.5. Other POSIX related changes

```
- A lot of character sets are supported now via a call to setlocale().
  The setting of the environment variables $LANG, $LC_ALL or $LC_CTYPE will
  be used.  For instance, setting $LANG to "de_DE.ISO-8859-15" before
  starting a Cygwin session will use the ISO-8859-15 character set in
  the entire session.  UTF-8 is supported as well, as in "en_US.UTF-8".

  The full list of supported character sets: "ASCII", "ISO-8859-x" with x
  in 1-16, except 12, "UTF-8", Windows codepages "CPxxx", with xxx in
  (437, 720, 737, 775, 850, 852, 855, 857, 858, 862, 866, 874, 1125,
  1250, 1251, 1252, 1253, 1254, 1255, 1256, 1257, 1258), "KOI8-R", "KOI8-U",
  "SJIS", "GBK", "eucJP", "eucKR", and "Big5".  The leading language and
  territory part (en_US, for instance) is not used by Cygwin yet, but is
  required for POSIX compatibility.

- Allow multiple concurrent read locks per thread for pthread_rwlock_t.

- Implement pthread_kill(thread, 0) as per POSIX.

- New API for POSIX IPC:
  Named semaphores: sem_open, sem_close, sem_unlink.
  Message queues: mq_open, mq_getattr, mq_setattr, mq_notify, mq_send,
  mq_timedsend, mq_receive, mq_timedreceive, mq_close, mq_unlink.
  Shared memory: shm_open, shm_unlink.
```

- Only declare expected functions in <strings.h>, don't include <string.h> from here.

- Support for WCONTINUED, WIFCONTINUED() added to waitpid and wait4.

- New APIs: _Exit, confstr, insque, remque, sys_sigabbrev,
  posix_madvise, posix_memalign, reallocf,
  exp10, exp10f, pow10, pow10f, lrint, lrintf, rint, rintf, llrint, llrintf,
  llrintl, lrintl, rintl,
  mbsnrtowcs, strcasestr, stpcpy, stpncpy, wcpcpy, wcpncpy, wcsnlen,
  wcsnrtombs, wcsftime, wcstod, wcstof, wcstoimax, wcstok, wcstol, wcstoll,
  wcstoul, wcstoull, wcstoumax, wcsxfrm, wcscasecmp, wcsncasecmp,
  fgetwc, fgetws, fputwc, fputws, fwide, getwc, getwchar, putwc, putwchar,
  ungetwc, asnprintf, dprintf, vasnprintf, vdprintf,
  wprintf, fwprintf, swprintf, vwprintf, vfwprintf, vswprintf,
  wscanf, fwscanf, swscanf, vwscanf, vfwscanf, vswscanf.

1.7.6. Security related changes

- Getting a domain user's groups is hopefully more bulletproof now.

- Cygwin now comes with a real LSA authentication package. This must be manually installed by a privileged user using the /bin/cyglsa-config script. The advantages and disadvantages are noted in http://cygwin.com/ml/cygwin-developers/2006-11/msg00000.html

- Cygwin now allows storage and use of user passwords in a hidden area of the registry. This is tried first when Cygwin is called by privileged processes to switch the user context. This allows, for instance, ssh public key sessions with full network credentials to access shares on other machines.

- New options have been added to the mkpasswd and mkgroup tools to ease use in multi-machine and multi-domain environments. The existing options have a slightly changed behaviour.

1.7.7. Miscellaneous

- New ldd utility, similar to Linux.

- New link libraries libdl.a, libresolve.a, librt.a.

- Fallout from the long path names: If the current working directory is longer than 260 bytes, or if the current working directory is a virtual path (like /proc, /cygdrive, //server), don't call native Win32 programs since they don't understand these paths.

- On the first usage of a DOS path (C:\foo, \\foo\bar), the Cygwin DLL

```
  emits a scary warning that DOS paths shouldn't be used.  This warning
  may be disabled via the new CYGWIN=nodosfilewarning setting.

- The CYGWIN environment variable option "server" has been removed.
  Cygwin automatically uses cygserver if it's available.

- Allow environment of arbitrary size instead of a maximum of 32K.

- Don't force uppercase environment when started from a non-Cygwin process.
  Except for certain Windows and POSIX variables which are always uppercased,
  preserve environment case.  Switch back to old behaviour with the new
  CYGWIN=upcaseenv setting.

- Detect and report a missing DLL on process startup.

- Add /proc/registry32 and /proc/registry64 paths to access 32 bit and
  64 bit registry on 64 bit systems.

- Add the ability to distinguish registry keys and registry values with the
  same name in the same registry subtree.  The key is called "foo" and the
  value will be called "foo%val" in this case.

- Align /proc/cpuinfo more closly to Linux content.

- Add /proc/$PID/mounts entries and a symlink /proc/mounts pointing to
  /proc/self/mounts as on Linux.

- Optimized strstr and memmem implementation.

- Remove backwards compatibility with old signal masks.  (Some *very* old
  programs which use signal masks may no longer work correctly).

- Cygwin now exports wrapper functions for libstdc++ operators new and
  delete, to support the toolchain in implementing full C++ standards
  conformance when working with shared libraries.
```

Chapter 2. Setting Up Cygwin

2.1. Internet Setup

To install the Cygwin net release, go to http://cygwin.com/ and click on "Install Cygwin Now!" (http://cygwin.com/setup.exe). This will download a GUI installer called **setup.exe** which can be run to download a complete cygwin installation via the internet. Follow the instructions on each screen to install Cygwin.

The **setup.exe** installer is designed to be easy for new users to understand while remaining flexible for the experienced. The volunteer development team is constantly working on **setup.exe**; before requesting a new feature, check the wishlist in the CVS `README` (http://sourceware.org/cgi-bin/cvsweb.cgi/setup/README?cvsroot=cygwin-apps&rev=2). It may already be present in the CVS version!

Since the default value for each option is the logical choice for most installations, you can get a working minimal Cygwin environment installed by simply clicking the `Next` button at each page. The only exception to this is choosing a Cygwin mirror, which you can choose by experimenting with those listed at http://cygwin.com/mirrors.html (http://cygwin.com/mirrors.html). For more details about each of page of the **setup.exe** installation, read on below. Please note that this guide assumes that you have a basic understanding of Unix (or a Unix-like OS). If you are new to Unix, you will also want to make use of other resources (http://www.google.com/search?q=new+to+unix).

2.1.1. Download Source

Cygwin uses packages to manage installing various software. When the default `Install from Internet` option is chosen, **setup.exe** creates a local directory to store the packages before actually installing the contents. `Download from Internet` performs only the first part (storing the packages locally), while `Install from Local Directory` performs only the second (installing the contents of the packages).

The `Download from Internet` option is mainly for creating a base Cygwin package tree on one computer for installation on several machines with `Install from Local Directory`; copy the entire local package tree to another machine with the directory tree intact. For example, you might create a `C:\cache\` directory and place **setup.exe** in it. Run **setup.exe** to `Install from Internet` or `Download from Internet`, then copy the whole `C:\cache\` to each machine and instead choose `Install from Local Directory`. Unfortunately **setup.exe** does not yet support unattended installs.

Though this provides some basic mirroring functionality, if you are managing a large Cygwin installation, to keep up to date we recommend using a mirroring tool such as **wget**. A helpful user on the

Cygwin mailing list created a simple demonstration script to accomplish this; search the list for **mkcygwget** for ideas.

2.1.2. Selecting an Install Directory

The `Root Directory` for Cygwin (default `C:\cygwin`) will become `/` within your Cygwin installation. You must have write access to the parent directory, and any ACLs on the parent directory will determine access to installed files.

The `Install For` options of `All Users` or `Just Me` should always be left on the default `All Users`, unless you do not have write access to `HKEY_LOCAL_MACHINE` in the registry or the All Users Start Menu. This is true even if you are the only user planning to use Cygwin on the machine. Selecting `Just Me` will cause problems for programs such as **crond** and **sshd**. If you do not have the necessary permissions, but still want to use these programs, consult the Cygwin mailing list archives about others' experiences.

The `Default Text File Type` should be left on `Unix` (that is, `\n`) unless you have a very good reason to switch it to `DOS` (that is, `\r\n`).

2.1.3. Local Package Directory

The `Local Package Directory` is the cache where **setup.exe** stores the packages before they are installed. The cache must not be the same folder as the Cygwin root. Within the cache, a separate directory is created for each Cygwin mirror, which allows **setup.exe** to use multiple mirrors and custom packages. After installing Cygwin, the cache is no longer necessary, but you may want to retain the packages as backups, for installing Cygwin to another system, or in case you need to reinstall a package.

2.1.4. Connection Method

The `Direct Connection` method of downloading will directly download the packages, while the IE5 method will leverage your IE5 cache for performance. If your organisation uses a proxy server or auto-configuration scripts, the IE5 method also uses these settings. If you have a proxy server, you can manually type it into the `Use Proxy` section. Unfortunately, **setup.exe** does not currently support password authorization for proxy servers.

2.1.5. Choosing Mirrors

Since there is no way of knowing from where you will be downloading Cygwin, you need to choose at least one mirror site. Cygwin mirrors are geographically distributed around the world; check the list at http://cygwin.com/mirrors.html to find one near you. You can select multiple mirrors by holding down

`CTRL` and clicking on each one. If you have the URL of an unlisted mirror (for example, if your organization has an internal Cygwin mirror) you can add it.

2.1.6. Choosing Packages

For each selected mirror site, **setup.exe** downloads a small text file called `setup.bz2` that contains a list of packages available from that site along with some basic information about each package which **setup.exe** parses and uses to create the chooser window. For details about the format of this file, see the setup.exe homepage (http://sourceware.org/cygwin-apps/setup.html).

The chooser is the most complex part of **setup.exe**. Packages are grouped into categories, and one package may belong to multiple categories (assigned by the volunteer package maintainer). Each package can be found under any of those categories in the heirarchial chooser view. By default, **setup.exe** will install only the packages in the `Base` category and their dependencies, resulting in a minimal Cygwin installation. However, this will not include many commonly used tools such as **gcc** (which you will find in the `Devel` category). Since **setup.exe** automatically selects dependencies, be careful not to unselect any required packages. In particular, everything in the `Base` category is required.

You can change **setup.exe**'s view style, which is helpful if you know the name of a package you want to install but not which category it is in. Click on the `View` button and it will rotate between `Category` (the default), `Full` (all packages), and `Partial` (only packages to be upgraded). If you are familiar with Unix, you will probably want to at least glance through the `Full` listing for your favorite tools.

Once you have an existing Cygwin installation, the **setup.exe** chooser is also used to manage your Cygwin installation. Information on installed packages is kept in the `/etc/setup/` directory of your Cygwin installation; if **setup.exe** cannot find this directory it will act as if you have no Cygwin installation. If **setup.exe** finds a newer version of an installed package available, it will automatically mark it to be upgraded. To `Uninstall`, `Reinstall`, or get the `Source` for an existing package, click on `Keep` to toggle it. Also, to avoid the need to reboot after upgrading, make sure to close all Cygwin windows and stop all Cygwin processes before **setup.exe** begins to install the upgraded package.

The final feature of the **setup.exe** chooser is for `Previous` and `Experimental` packages. By default the chooser shows only the current version of each package, though mirrors have at least one previous version and occasionally there is a testing or beta version of a package available. To see these package, click on the `Prev` or `Exp` radio button. Be warned, however, that the next time you run **setup.exe** it will try to replace old or experimental versions with the current stable version.

2.1.7. Download and Installation Progress

First, **setup.exe** will download all selected packages to the local directory chosen earlier. Before installing, **setup.exe** performs a checksum on each package. If the local directory is a slow medium (such as a network drive) this can take a long time. During the download and installation, **setup.exe** shows progress bars for the current task and total remaining disk space.

2.1.8. Icons

You may choose to install shortcuts on the Desktop and/or Start Menu to start a `bash` shell. If you prefer to use a different shell or the native Windows version of `rxvt`, you can use these shortcuts as a guide to creating your own.

2.1.9. Post-Install Scripts

Last of all, **setup.exe** will run any post-install scripts to finish correctly setting up installed packages. Since each script is run separately, several windows may pop up. If you are interested in what is being done, see the Cygwin Package Contributor's Guide at http://cygwin.com/setup.html When the last post-install script is completed, **setup.exe** will display a box announcing the completion. A few packages, such as the OpenSSH server, require some manual site-specific configuration. Relevant documentation can be found in the `/usr/doc/Cygwin/` or `/usr/share/doc/Cygwin/` directory.

2.1.10. Troubleshooting

Unfortunately, the complex setup process means that odd problems can occur. If you're having trouble downloading packages, it may be network congestion, so try a different mirror and/or a different protocol (i.e., HTTP instead of FTP). If you notice something is not working after running setup, you can check the **setup.exe** log file at `/var/log/setup.log.full`. Make a backup of this file before running **setup.exe** again, and follow the steps for Reporting Problems with Cygwin (http://cygwin.com/problems.html).

2.2. Environment Variables

You may wish to specify settings of several important environment variables that affect Cygwin's operation. Some of these settings need to be in effect prior to launching the initial Cygwin session (before starting your bash shell, for instance), and are, consequentially, best placed in a .bat file. An initial file is named Cygwin.bat and is created in the Cygwin root directory that you specified during setup. Note that the "Cygwin" option of the Start Menu points to Cygwin.bat. Edit Cygwin.bat to your liking or create your own .bat files to start Cygwin processes.

The CYGWIN variable is used to configure many global settings for the Cygwin runtime system. Initially you can leave CYGWIN unset or set it to `tty` (e.g. to support job control with ^Z etc...) using a syntax like this in the DOS shell, before launching bash.

```
C:\> set CYGWIN=tty notitle glob
```

Locale support is controlled by the LANG and LC_xxx environment variables. You can set all of them but Cygwin itself only honors the variables LC_ALL, LC_CTYPE, and LANG, in this order, according to the POSIX standard. The first one found rules. For a more detailed description see Section 2.4.

The PATH environment variable is used by Cygwin applications as a list of directories to search for executable files to run. This environment variable is converted from Windows format (e.g. `C:\Windows\system32;C:\Windows`) to UNIX format (e.g., `/cygdrive/c/Windows/system32:/cygdrive/c/Windows`) when a Cygwin process first starts. Set it so that it contains at least the `x:\cygwin\bin` directory where "`x:\cygwin` is the "root" of your cygwin installation if you wish to use cygwin tools outside of bash. This is usually done by the batch file you're starting your shell with.

The HOME environment variable is used by many programs to determine the location of your home directory and we recommend that it be defined. This environment variable is also converted from Windows format when a Cygwin process first starts. It's usually set in the shell profile scripts in the /etc directory.

The TERM environment variable specifies your terminal type. It is automatically set to `cygwin` if you have not set it to something else.

The LD_LIBRARY_PATH environment variable is used by the Cygwin function `dlopen ()` as a list of directories to search for .dll files to load. This environment variable is converted from Windows format to UNIX format when a Cygwin process first starts. Most Cygwin applications do not make use of the `dlopen ()` call and do not need this variable.

2.3. Changing Cygwin's Maximum Memory

Cygwin's heap is extensible. However, it does start out at a fixed size and attempts to extend it may run into memory which has been previously allocated by Windows. In some cases, this problem can be solved by adding an entry in the either the `HKEY_LOCAL_MACHINE` (to change the limit for all users) or `HKEY_CURRENT_USER` (for just the current user) section of the registry.

Add the `DWORD` value `heap_chunk_in_mb` and set it to the desired memory limit in decimal MB. It is preferred to do this in Cygwin using the **regtool** program included in the Cygwin package. (For more information about **regtool** or the other Cygwin utilities, see Section 3.7 or use the `--help` option of each util.) You should always be careful when using **regtool** since damaging your system registry can result in an unusable system. This example sets memory limit to 1024 MB:

```
regtool -i set /HKLM/Software/Cygwin/heap_chunk_in_mb 1024
regtool -v list /HKLM/Software/Cygwin
```

Exit all running Cygwin processes and restart them. Memory can be allocated up to the size of the system swap space minus any the size of any running processes. The system swap should be at least as large as the physically installed RAM and can be modified under the System category of the Control Panel.

Here is a small program written by DJ Delorie that tests the memory allocation limit on your system:

```
main()
{
  unsigned int bit=0x40000000, sum=0;
  char *x;

  while (bit > 4096)
  {
    x = malloc(bit);
    if (x)
    sum += bit;
    bit >>= 1;
  }
  printf("%08x bytes (%.1fMb)\n", sum, sum/1024.0/1024.0);
  return 0;
}
```

You can compile this program using:

```
gcc max_memory.c -o max_memory.exe
```

Run the program and it will output the maximum amount of allocatable memory.

2.4. Internationalization

2.4.1. Overview

Internationalization support is controlled by the LANG and LC_xxx environment variables. You can set all of them but Cygwin itself only honors the variables LC_ALL, LC_CTYPE, and LANG, in this order, according to the POSIX standard. The content of these variables should follow the POSIX standard for a locale specifier. The correct form of a locale specifier is

```
  language[[_TERRITORY][.charset][@modifier]]
```

"language" is a lowercase two character string per ISO 639-1, "TERRITORY" is an uppercase two character string per ISO 3166, charset is one of a list of supported character sets, and the modifier doesn't matter here (though it might for some applications). If you're interested in the exact description, you can find it in the online publication of the POSIX manual pages on the homepage of the Open Group (http://www.opengroup.org/).

Typical locale specifiers are

```
"de_CH"    language = German, territory = Switzerland, default charset
"fr_FR.UTF-8"    language = french, territory = France, charset = UTF-8
"ko_KR.eucKR"    language = korean, territory = South Korea, charset = eucKR
```

At application startup, the application's locale is set to the default "C" or "POSIX" locale. Under Cygwin, this locale defaults to the UTF-8 character set. If you want to stick to the "C" locale and only change to another charset, you can define this by setting one of the locale environment variables to "C.charset". For instance

```
"C.ISO-8859-1"
```

Windows uses the UTF-16 charset exclusively to store the names of any object used by the Operating System. This is especially important with filenames. Cygwin uses the setting of the locale environment variables LC_ALL, LC_CTYPE, and LANG, to determine how to convert Windows filenames from their UTF-16 representation to the singlebyte or multibyte character set used by Cygwin.

The setting of the locale environment variables at process startup is effective for Cygwin's internal conversions to and from the Windows UTF-16 object names for the entire lifetime of the current process. Changing the environment variables to another value changes the way filenames are converted in subsequently started child processes, but not within the same process.

However, even if one of the locale environment variables is set to some other value than "C", this does *only* affect how Cygwin itself converts filenames. As the POSIX standard requires, it's the applications responsibility to activate that locale for its own purpose, typically by using the call

```
setlocale (LC_ALL, "");
```

early in the application code.

Right now the language and territory, as well as the modifier, are not important to Cygwin, except to fix a single problem. There's a class of characters in the Unicode character set, called the "CJK Ambiguous Width Character set". For these characters the width returned by the wcwidth/wcswidth function is usually 1. This is often a problem in East-Asian languages, which historically use character sets in which these characters have a width of 2. Kind of explains why they are called "ambiguous"...

The problem has been fixed like this. wcwidth/wcswidth usually return 1 as the width of these characters. However, if the language is specifed as "ja" (Japanese), "ko" (Korean), or "zh" (Chinese), wcwidth returns 2 for these characters. Unfortunately this isn't correct in all circumstances, so the user can specify the modifier "@cjknarrow", which modifies the behaviour of wcwidth/wcswidth to return 1 for the ambiguous width characters to return 1 even in those languages.

Other than that, the only important part so far is the character set. How does that work?

2.4.2. How to set the locale

- The default locale is the "C" or "POSIX" locale. Under Cygwin this locale defaults to the UTF-8 character set.
- Assume that you've set one of the aforementioned environment variables to some valid POSIX locale value, other than "C" and "POSIX". Assume further that you're living in Japan. You might want to use the language code "ja" and the territory "JP", thus setting, say, LANG to "ja_JP". You didn't set a character set, so what will Cygwin use now? Easy! It will use the default Windows ANSI codepage of your system, if it's supported by Cygwin. Hopefully Cygwin supports all relevant default ANSI codepages...

  Note: For a list of supported character sets, see Section 2.4.6

- You don't want to use the default Windows codepage as character set? In that case you have to specify the charset explicitly. For instance, assume you're from Italy and don't want to use the Italian default Windows ANSI codepage 1252, but the more portable ISO-8859-15 character set. What you can do, for instance, is to set the LANG variable in the `C:\cygwin\Cygwin.bat` file which is the batch file to start a Cygwin session from the "Cygwin" desktop shortcut.

  ```
  @echo off

  C:
  chdir C:\cygwin\bin
  set LANG=it_IT.ISO-8859-15
  bash --login -i
  ```

- Last, but not least, most singlebyte or doublebyte charsets have a big disadvantage. Windows filesystems use the Unicode character set in the UTF-16 encoding to store filename information. Not all characters from the Unicode character set are available in a singlebyte or doublebyte charset. While Cygwin has a workaround to access files with unusual characters (see Section 3.4.4), a better workaround is to use always the UTF-8 character set.i

  UTF-8 is the only multibyte character set which can represent every Unicode character.

  ```
  set LANG=es_MX.UTF-8
  ```

  For a description of the Unicode standard, see the homepage of the Unicode Consortium (http://www.unicode.org/).

2.4.3. The Windows Console character set

Most of the time the Windows console is used to run Cygwin applications. While terminal emulations like **xterm** or **mintty** have a distinct way to set the character set used for in- and output, the Windows console hasn't such a way, since it's not an application in its own right.

This problem is solved in Cygwin as follows. When a Cygwin process is started in a Windows console (either explicitly from cmd.exe, or implicitly by, for instance, clicking on the Cygwin desktop icon, or running the Cygwin.bat file), the Console character set is determined by the setting of the aforementioned internationalization environment variables, the same way as described in Section 2.4.2.

What is that good for? Why not switch the console character set with the applications requirements? After all, the application knows if it uses localization or not. However, what if a non-localized application calls a remote application which itself is localized? This can happen with **ssh** or **rlogin**. Both commands don't have and don't need localization and they never call `setlocale`. Setting one of the internationalization environment variable to the same charset as the remote machine before starting **ssh** or **rlogin** fixes that problem.

2.4.4. Potential Problems when using Locales

You can set the above internationalization variables not only in `Cygwin.bat` or in the Windows environment, but also in your Cygwin shell on the fly, even switch to yet another character set, and yet another. In bash for instance:

```
bash$ export LC_CTYPE="nl_BE.UTF-8"
```

However, here's a problem. At the start of the first Cygwin process in a session, the Windows environment is converted from UTF-16 to UTF-8. The environment is another of the system objects stored in UTF-16 in Windows.

As long as the environment only contains ASCII characters, this is no problem at all. But if it contains native characters, and you're planning to use, say, GBK, the environment will result in invalid characters in the GBK charset. This would be especially a problem in variables like PATH. To circumvent the worst problems, Cygwin converts the PATH environment variable to the charset set in the environment, if it's different from the UTF-8 charset.

> **Note:** Per POSIX, the name of an environment variable should only consist of valid ASCII characters, and only of uppercase letters, digits, and the underscore for maximum portablilty.

Symbolic links, too, may pose a problem when switching charsets on the fly. A symbolic link contains the filename of the target file the symlink points to. When a symlink had been created with older versions of Cygwin, the current ANSI or OEM character set had been used to store the target filename, dependent on the old CYGWIN environment variable setting codepage (see Section 3.5.2. If the target filename contains non-ASCII characters and you use another character set than your default ANSI/OEM charset, the target filename of the symlink is now potentially an invalid character sequence in the new character set. This behaviour is not different from the behaviour in other Operating Systems. So, if you suddenly can't access a symlink anymore which worked all these years before, maybe it's because you switched to another character set. This doesn't occur with symlinks created with Cygwin 1.7 or later.

Another problem you might encounter is that older versions of Windows did not install all charsets by default. If you are running Windows XP or older, you can open the "Regional and Language Options" portion of the Control Panel, select the "Advanced" tab, and select entries from the "Code page conversion tables" list. The following entries are useful to cygwin: 932/SJIS, 936/GBK, 949/EUC-KR, 950/Big5, 20932/EUC-JP.

2.4.5. What does not work?

Except for LC_ALL, LC_CTYPE, and LANG, all other LC_xxx environment variables, LC_COLLATE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME, are ignored right now. This means, while Cygwin supports different character sets, it does *not* support real localization so far. There's no support for locale-specific monetary symbols, for a decimalpoint other than '.', no support for native time formats, and no support for native language sorting orders.

Cygwin's internationalization support is work in progress and we would be glad for coding help in this area.

2.4.6. List of supported character sets

Last but not least, here's the list of currently supported character sets. The left-hand expression is the name of the charset, as you would use it in the internationalization environment variables as outlined above. Note that charset specifiers are case-insensitive. `EUCJP` is equivalent to `eucJP` or `eUcJp`. Writing the charset in the exact case as given in the list below is a good convention, though.

The right-hand side is the number of the equivalent Windows codepage as well as the Windows name of the codepage. They are only noted here for reference. Don't try to use the bare codepage number or the Windows name of the codepage as charset in locale specifiers, unless they happen to be identical with the left-hand side. Especially in case of the "CPxxx" style charsets, always use them with the trailing "CP".

This works:

```
set LC_ALL=en_US.CP437
```

This does *not* work:

```
set LC_ALL=en_US.437
```

You can find a full list of Windows codepages on the Microsoft MSDN page Code Page Identifiers (http://msdn.microsoft.com/en-us/library/dd317756(VS.85).aspx).

```
Charset                 Codepage

CP437                     437 (OEM United States)
CP720                     720 (DOS Arabic)
CP737                     737 (OEM Greek)
CP775                     775 (OEM Baltic)
CP850                     850 (OEM Latin 1, Western European)
CP852                     852 (OEM Latin 2, Central European)
CP855                     855 (OEM Cyrillic)
CP857                     857 (OEM Turkish)
CP858                     858 (OEM Latin 1 + Euro Symbol)
CP862                     862 (OEM Hebrew)
CP866                     866 (OEM Russian)
CP874                     874 (ANSI/OEM Thai)
CP1125                   1125 (OEM Ukraine)
CP1250                   1250 (ANSI Central European)
CP1251                   1251 (ANSI Cyrillic)
CP1252                   1252 (ANSI Latin 1, Western European)
CP1253                   1253 (ANSI Greek)
CP1254                   1254 (ANSI Turkish)
CP1255                   1255 (ANSI Hebrew)
CP1256                   1256 (ANSI Arabic)
CP1257                   1257 (ANSI Baltic)
CP1258                   1258 (ANSI/OEM Vietnamese)

ISO-8859-1              28591 (ISO-8859-1)
ISO-8859-2              28592 (ISO-8859-2)
ISO-8859-3              28593 (ISO-8859-3)
ISO-8859-4              28594 (ISO-8859-4)
ISO-8859-5              28595 (ISO-8859-5)
ISO-8859-6              28596 (ISO-8859-6)
ISO-8859-7              28597 (ISO-8859-7)
ISO-8859-8              28598 (ISO-8859-8)
ISO-8859-9              28599 (ISO-8859-9)
ISO-8859-10               -   (not available)
ISO-8859-11               -   (not available)
ISO-8859-13             28603 (ISO-8859-13)
ISO-8859-14               -   (not available)
ISO-8859-15             28605 (ISO-8859-15)
ISO-8859-16               -   (not available)

KOI8-R                  20866 (KOI8-R Russian Cyrillic)
KOI8-U                  21866 (KOI8-U Ukrainian Cyrillic)
SJIS                      932 (ANSI/OEM Japanese)
GBK                       936 (ANSI/OEM Simplified Chinese)
Big5                      950 (ANSI/OEM Traditional Chinese)
eucJP                   20932 (EUC Japanese)
eucKR                     949 (EUC Korean)

UTF-8 or UTF8           65001 (UTF-8)
```

2.5. Using Windows security in Cygwin

This section discusses how the Windows security model is utilized in Cygwin to implement POSIX-like permissions, as well as how the Windows authentication model is used to allow cygwin applications to switch users in a POSIX-like fashion.

The setting of POSIX-like file and directory permissions is controlled by the mount option `(no)acl` which is set to `acl` by default.

We start with a short overview. Note that this overview must be necessarily short. If you want to learn more about the Windows security model, see the Access Control (http://msdn.microsoft.com/en-us/library/aa374860(VS.85).aspx) article in MSDN documentation.

POSIX concepts and specificially the POSIX security model are not discussed here, but assumed to be understood by the reader. If you don't know the POSIX security model, search the web for beginner documentation.

2.5.1. Overview

In the Windows security model, almost any "object" is securable. "Objects" are files, processes, threads, semaphores, etc.

Every object has a data structure attached, called a "security descriptor" (SD). The SD contains all information necessary to control who can access an object, and to determine what they are allowed to do to or with it. The SD of an object consists of five parts:

- Flags which control several aspects of this SD. This is not discussed here.
- The SID of the object owner.
- The SID of the object owner group.
- A list of "Access Control Entries" (ACE), called the "Discretionary Access Control List" (DACL).
- Another list of ACEs, called the "Security Access Control List" (SACL), which doesn't matter for our purpose. We ignore it here.

Every ACE contains a so-called "Security IDentifier" (SID) and other stuff which is explained a bit later. Let's talk about the SID first.

A SID is a unique identifier for users, groups, computers and Active Directory (AD) domains. SIDs are basically comparable to POSIX user ids (UIDs) and group ids (GIDs), but are more complicated because they are unique across multiple machines or domains. A SID is a structure of multiple numerical values. There's a convenient convention to type SIDs, as a string of numerical fields separated by hyphen characters. Here's an example:

SID of a machine "foo":

```
S-1-5-21-165875785-1005667432-441284377
```

SID of a user "johndoe" of the system "foo":

```
S-1-5-21-165875785-1005667432-441284377-1023
```

The first field is always "S", which is just a notational convention to show that this is a SID. The second field is the version number of the SID structure, So far there exists only one version of SIDs, so this field is always 1. The third and fourth fields represent the "authority" which can be thought of as a type or category of SIDs. There are a couple of builtin accounts and accounts with very special meaning which have certain well known values in these third and fourth fields. However, computer and domain SIDs always start with "S-1-5-21". The next three fields, all 32 bit values, represent the unique 96 bit identifier of the computer system. This is a hopefully unique value all over the world, but in practice it's sufficient if the computer SIDs are unique within a single Windows network.

As you can see in the above example, SIDs of users (and groups) are identical to the computer SID, except for an additional part, the so-called "relative identifier" (RID). So the SID of a user is always uniquely attached to the system on which the account has been generated.

It's a bit different in domains. The domain has its own SID, and that SID is identical to the SID of the first domain controller, on which the domain is created. Domain user SIDs look exactly like the computer user SIDs, the leading part is just the domain SID and the RID is created when the user is created.

Ok, consider you created a new domain "bar" on some new domain controller and you would like to create a domain account "johndoe":

SID of a domain "bar.local":

```
S-1-5-21-186985262-1144665072-740312968
```

SID of a user "johndoe" in the domain "bar.local":

```
S-1-5-21-186985262-1144665072-740312968-1207
```

So you now have two accounts called johndoe, one account created on the machine "foo", one created in the domain "bar.local". Both have different SIDs and not even the RID is the same. How do the systems know it's the same account? After all, the name is the same, right? The answer is, these accounts are **not** identical. All machines on the network will treat these SIDs as identifying two separate accounts. One is "FOO\johndoe", the other one is "BAR\johndoe" or "johndoe@bar.local". Different SID, different account. Full stop.

The last part of the SID, the so called "Relative IDentifier" (RID), is by default used as UID and/or GID under Cygwin when you create the `/etc/passwd` and `/etc/group` files using the **mkpasswd** and **mkgroup** tools. Domain account UIDs and GIDs are offset by 10000 by default which might be a bit low for very big organizations. Fortunately there's an option in both tools to change the offset...

Do you still remember the SIDs with special meaning? In offical notation they are called "well-known SIDs". For example, POSIX has no GID for the group of "all users" or "world" or "others". The last three rwx bits in a unix-style permission value just represent the permissions for "everyone who is not the owner or is member of the owning group". Windows has a SID for these poor souls, the "Everyone" SID. Other well-known SIDs represent circumstances under which a process is running, rather than actual users or groups. Here are a few examples for well-known SIDs:

```
Everyone                        S-1-1-0    Simply everyone...
Batch                           S-1-5-3    Processes started via the task
        scheduler are member of this group.
Interactive   S-1-5-4    Only processes of users which are
        logged in via an interactive
        session are members here.
Authenticated Users             S-1-5-11   Users which have gone through
                                           the authentication process and
        survived.  Anonymously accessing
        users are not incuded here.
SYSTEM                          S-1-5-18   A special account which has all
        kinds of dangerous rights, sort of
        an uber-root account.
```

For a full list please refer to the MSDN document Well-known SIDs (http://msdn.microsoft.com/en-us/library/aa379649.aspx). The Cygwin package called "csih" provides a tool, /usr/lib/csih/getAccountName.exe, which can be used to print the (possibly localized) name for the various well-known SIDS.

Naturally, well-known SIDs are the same on each machine, so they are not unique to a machine or domain. They have the same meaning across the Windows network.

Additionally, there are a couple of well-known builtin groups, which have the same SID on every machine and which have certain user rights by default:

```
administrators                  S-1-5-32-544
users                           S-1-5-32-545
guests                          S-1-5-32-546
...
```

For instance, every account is usually member in the "Users" group. All administrator accounts are member of the "Administrators" group. That's all about it as far as single machines are involved. In a domain environment it's a bit more tricky. Since these SIDs are not unique to a machine, every domain user and every domain group can be a member of these well known groups. Consider the domain group "Domain Admins". This group is by default in the "Administrators" group. Let's assume the above computer called "foo" is a member machine of the domain "bar.local". If you stick the user

"BAR\johndoe" into the group "Domain Admins", this guy will automatically be a member of the administrators group on "foo" when logging on to "foo". Neat, isn't it?

Back to ACE and ACL. POSIX is able to create three different permissions, the permissions for the owner, for the group and for the world. In contrast the Windows ACL has a potentially infinite number of members... as long as they fit into 64K. Every member is an ACE. ACE consist of three parts:

- The type of the ACE (allow ACE or deny ACE).
- Permission bits, 32 of them.
- The SID for which the permissions are allowed or denied.

The two (for us) important types of ACEs are the "access allowed ACE" and the "access denied ACE". As the names imply, the allow ACE tells the system to allow the given permissions to the SID, the deny ACE results in denying the specific permission bits.

The possible permissions on objects are more detailed than in POSIX. For example, the permission to delete an object is different from the permission to change object data, and even changing object data can be separated into different permission bits for different kind of data. But there's a problem with the definition of a "correct" ACL which disallows mapping of certain POSIX permissions cleanly. See Section 2.5.4.

POSIX is able to create only three different permissions? Not quite. Newer operating systems and file systems on POSIX systems also provide access control lists. Two different APIs exist for accessing these ACLs, the Solaris API and the POSIX API. Cygwin implements the Solaris API to access Windows ACLs in a Unixy way. At the time of writing this document, the Cygwin implementation of the Solaris API isn't quite up to speed. For instance, it doesn't handle access denied ACEs gracefully. So, use with care. Online man pages for the Solaris ACL API can be found on http://docs.sun.com.

2.5.2. File permissions

On NTFS and if the `noacl` mount option is not specified for a mount point, Cygwin sets file permissions as in POSIX. Basically this is done by defining a SD with the matching owner and group SIDs, and a DACL which contains ACEs for the owner, the group and for "Everyone", which represents what POSIX calls "others".

To use Windows security correctly, Cygwin depends on the files `/etc/passwd` and `/etc/group`. These files define the translation between the Cygwin uid/gid and the Windows SID. The SID is stored in the pw_gecos field in `/etc/passwd`, and in the gr_passwd field in `/etc/group`. Since the pw_gecos field can contain more information than just a SID, there are some rules for the layout. It's required that the SID is the last entry of the pw_gecos field, assuming that the entries in pw_gecos are comma-separated. The commands **mkpasswd** and **mkgroup** usually do this for you.

Another interesting entry in the pw_gecos field (which is also usually created by running **mkpasswd**) is the Windows user name entry. It takes the form "U-domain\username" and is sometimes used by services to authenticate a user. Logging in through **telnet** is a common scenario.

A typical snippet from `/etc/passwd`:

Example 2-1. /etc/passwd:

```
SYSTEM:*:18:544:,S-1-5-18::
Administrators:*:544:544:,S-1-5-32-544::
Administrator:unused:500:513:U-FOO\Administrator,S-1-5-21-790525478-115176313-839522115-500
corinna:unused:11001:11125:U-BAR\corinna,S-1-5-21-2913048732-1697188782-3448811101-1001:/ho
```

The SYSTEM entry is usually needed by services. The Administrators entry (Huh? A group in /etc/passwd?) is only here to allow **ls** and similar commands to print some file ownerships correctly. Windows doesn't care if the owner of a file is a user or a group. In older versions of Windows NT the default ownership for files created by an administrator account was set to the group Administrators instead of to the creating user account. This has changed, but you can still switch to this setting on newer systems. So it's convenient to have the Administrators group in `/etc/passwd`.

The really interesting entries are the next two. The Administrator entry is for the local administrator, the corinna entry matches the corinna account in the domain BAR. The information given in the pw_gecos field are all we need to exactly identify an account, and to have a two way translation, from Windows account name/SID to Cygwin account name uid and vice versa. Having this complete information allows us to choose a Cygwin user name and uid which doesn't have to match the Windows account at all. As long as the pw_gecos information is available, we're on the safe side:

Example 2-2. /etc/passwd, tweaked:

```
root:unused:0:513:U-FOO\Administrator,S-1-5-21-790525478-115176313-839522115-500:/home/Admi
thursday_next:unused:11001:11125:U-BAR\corinna,S-1-5-21-2913048732-1697188782-3448811101-10
```

The above `/etc/passwd` will still work fine. You can now login via **ssh** as the user "root", and Cygwin dutifully translates "root" into the Windows user "FOO\Administrator" and files owned by FOO\Administrator are shown to have the uid 0 when calling **ls -ln**. All you do you're actually doing as Administrator. Files created as root will be owned by FOO\Administrator. And the domain user BAR\corinna can now happily pretend to be Thursday Next, but will wake up sooner or later finding out she's still actually the domain user BAR\corinna...

Do I have to mention that you can also rename groups in `/etc/group`? As long as the SID is present and correct, all is well. This allows you to, for instance, rename the "Administrators" group to "root" as well:

Example 2-3. /etc/group, tweaked:

```
root:S-1-5-32-544:544:
```

Last but not least, you can also change the primary group of a user in `/etc/passwd`. The only requirement is that the user is actually a member of the new primary group in Windows. For instance, normal users in a domain environment are members in the group "Domain Users", which in turn belongs to the well-known group "Users". So, if it's more convenient in your environment for the user's primary group to be "Users", just set the user's primary group in `/etc/passwd` to the Cygwin uid of "Users" (see in `/etc/group`, default 545) and let the user create files with a default group ownership of "Users".

> **Note:** If you wish to make these kind of changes to /etc/passwd and /etc/group, do so only if you feel comfortable with the concepts. Otherwise, do not be surprised if things break in either subtle or surprising ways! If you do screw things up, revert to copies of `/etc/passwd` and `/etc/group` files created by **mkpasswd** and **mkgroup**. (Make backup copies of these files before modifying them.) Especially, don't change the UID or the name of the user SYSTEM. It may mostly work, but some Cygwin applications running as a local service under that account could suddenly start behaving strangely.

2.5.3. Special values of user and group ids

If the current user is not present in `/etc/passwd`, that user's uid is set to a special value of 400. The user name for the current user will always be shown correctly. If another user (or a Windows group, treated as a user) is not present in `/etc/passwd`, the uid of that user will have a special value of -1 (which would be shown by **ls** as 65535). The user name shown in this case will be '????????'.

If the current user is not present in `/etc/passwd`, that user's login gid is set to a special value of 401. The gid 401 is shown as 'mkpasswd', indicating the command that should be run to alleviate the situation.

If another user is not present in `/etc/passwd`, that user's login gid is set to a special value of -1. If the user is present in `/etc/passwd`, but that user's group is not in `/etc/group` and is not the login group of that user, the gid is set to a special value of -1. The name of this group (id -1) will be shown as '????????'.

If the current user is present in `/etc/passwd`, but that user's login group is not present in `/etc/group`, the group name will be shown as 'mkgroup', again indicating the appropriate command.

A special case is if the current user's primary group SID is noted in the user's `/etc/passwd` entry using another group id than the group entry of the same group SID in `/etc/group`. This should be noted and corrected. The group name printed in this case is 'passwd/group_GID_clash(PPP/GGG)', with PPP being the gid as noted in `/etc/passwd` and GGG the gid as noted in `/etc/group`.

To summarize:

- If the current user doesn't show up in `/etc/passwd`, it's *group* will be named 'mkpasswd'.
- Otherwise, if the login group of the current user isn't in `/etc/group`, it will be named 'mkgroup'.
- Otherwise a group not in `/etc/group` will be shown as '????????' and a user not in `/etc/passwd` will be shown as "????????".

- If different group ids are used for a group with the same SID, the group name is shown as 'passwd/group_GID_clash(PPP/GGG)' with PPP and GGG being the different group ids.

Note that, since the special user and group names are just indicators, nothing prevents you from actually having a user named 'mkpasswd' in `/etc/passwd` (or a group named 'mkgroup' in `/etc/group`). If you do that, however, be aware of the possible confusion.

2.5.4. The POSIX permission mapping leak

As promised earlier, here's the problem when trying to map the POSIX permission model onto the Windows permission model.

There's a leak in the definition of a "correct" ACL which disallows a certain POSIX permission setting. The official documentation explains in short the following:

- The requested permissions are checked against all ACEs of the user as well as all groups the user is member of. The permissions given in these user and groups access allowed ACEs are accumulated and the resulting set is the set of permissions of that user given for that object.
- The order of ACEs is important. The system reads them in sequence until either any single requested permission is denied or all requested permissions are granted. Reading stops when this condition is met. Later ACEs are not taken into account.
- All access denied ACEs **should** precede any access allowed ACE. ACLs following this rule are called "canonical"

Note that the last rule is a preference or a definition of correctness. It's not an absolute requirement. All Windows kernels will correctly deal with the ACL regardless of the order of allow and deny ACEs. The second rule is not modified to get the ACEs in the preferred order.

Unfortunately the security tab in the file properties dialog of the Windows NT4 explorer is completely unable to deal with access denied ACEs while the Windows 2000 and later properties dialog rearranges the order of the ACEs to canonical order before you can read them. Thank God, the sort order remains unchanged if one presses the Cancel button. But don't even **think** of pressing OK...

Canonical ACLs are unable to reflect each possible combination of POSIX permissions. Example:

```
rw-r-xrw-
```

Ok, so here's the first try to create a matching ACL, assuming the Windows permissions only have three bits, as their POSIX counterpart:

```
UserAllow:   110
GroupAllow:  101
OthersAllow: 110
```

Hmm, because of the accumulation of allow rights the user may execute because the group may execute.

Second try:

```
UserDeny:    001
GroupAllow:  101
OthersAllow: 110
```

Now the user may read and write but not execute. Better? No! Unfortunately the group may write now because others may write.

Third try:

```
UserDeny:    001
GroupDeny:   010
GroupAllow:  001
OthersAllow: 110
```

Now the group may not write as intended but unfortunately the user may not write anymore, either. How should this problem be solved? According to the canonical order a UserAllow has to follow the GroupDeny but it's easy to see that this can never be solved that way.

The only chance:

```
UserDeny:    001
UserAllow:   010
GroupDeny:   010
GroupAllow:  001
OthersAllow: 110
```

Again: This works on all existing versions of Windows NT, at the time of writing from at least NT4 up to Server 2008. Only the GUIs aren't able (or willing) to deal with that order.

2.5.5. Switching the user context

Since Windows XP, Windows users have been accustomed to the "Switch User" feature, which switches the entire desktop to another user while leaving the original user's desktop "suspended". Another Windows feature (since Windows 2000) is the "Run as..." context menu entry, which allows you to start an application using another user account when right-clicking on applications and shortcuts.

On POSIX systems, this operation can be performed by processes running under the privileged user accounts (usually the "root" user account) on a per-process basis. This is called "switching the user context" for that process, and is performed using the POSIX **setuid** and **seteuid** system calls.

While this sort of feature is available on Windows as well, Windows does not support the concept of these calls in a simple fashion. Switching the user context in Windows is generally a tricky process with lots of "behind the scenes" magic involved.

Windows uses so-called ‘access tokens’ to identify a user and its permissions. Usually the access token is created at logon time and then it’s attached to the starting process. Every new process within a session inherits the access token from its parent process. Every thread can get its own access token, which allows, for instance, to define threads with restricted permissions.

2.5.6. Switching the user context with password authentication

To switch the user context, the process has to request such an access token for the new user. This is typically done by calling the Win32 API function **LogonUser** with the user name and the user’s cleartext password as arguments. If the user exists and the password was specified correctly, the access token is returned and either used in **ImpersonateLoggedOnUser** to change the user context of the current thread, or in **CreateProcessAsUser** to change the user context of a spawned child process.

Later versions of Windows define new functions in this context and there are also functions to manipulate existing access tokens (usually only to restrict them). Windows Vista also adds subtokens which are attached to other access tokens which plays an important role in the UAC (User Access Control) facility of Vista and later. However, none of these extensions to the original concept are important for this documentation.

Back to this logon with password, how can this be used to implement **set(e)uid**? Well, it requires modification of the calling application. Two Cygwin functions have been introduced to support porting **setuid** applications which only require login with passwords. You only give Cygwin the right access token and then you can call **seteuid** or **setuid** as usual in POSIX applications. Porting such a **setuid** application is illustrated by a short example:

```
/* First include all needed cygwin stuff. */
#ifdef __CYGWIN__
#include <windows.h>
#include <sys/cygwin.h>
#endif

[...]

  struct passwd *user_pwd_entry = getpwnam (username);
  char *cleartext_password = getpass ("Password:");

[...]

#ifdef __CYGWIN__
  /* Patch the typical password test. */
```

```
  {
    HANDLE token;

    /* Try to get the access token from Windows. */
    token = cygwin_logon_user (user_pwd_entry, cleartext_password);
    if (token == INVALID_HANDLE_VALUE)
       error_exit;
    /* Inform Cygwin about the new impersonation token. */
    cygwin_set_impersonation_token (token);
    /* Cygwin is now able, to switch to that user context by setuid or seteuid calls. */
  }
#else
    /* Use standard method on non-Cygwin systems. */
    hashed_password = crypt (cleartext_password, salt);
    if (!user_pwd_entry ||
        strcmp (hashed_password, user_pwd_entry->pw_password))
      error_exit;
#endif /* CYGWIN */

[...]

  /* Everything else remains the same! */

  setegid (user_pwd_entry->pw_gid);
  seteuid (user_pwd_entry->pw_uid);
  execl ("/bin/sh", ...);
```

2.5.7. Switching the user context without password, Method 1: Create a token from scratch

An unfortunate aspect of the implementation of **set(e)uid** is the fact that the calling process requires the password of the user to which to switch. Applications such as **sshd** wishing to switch the user context after a successful public key authentication, or the **cron** application which, again, wants to switch the user without any authentication are stuck here. But there are other ways to get new user tokens.

One way is just to create a user token from scratch. This is accomplished by using an (officially undocumented) function on the NT function level. The NT function level is used to implement the Win32 level, and, as such is closer to the kernel than the Win32 level. The function of interest, **NtCreateToken**, allows you to specify user, groups, permissions and almost everything you need to create a user token, without the need to specify the user password. The only restriction for using this function is that the calling process needs the "Create a token object" user right, which only the SYSTEM user account has by default, and which is considered the most dangerous right a user can have on Windows systems.

That sounds good. We just start the servers which have to switch the user context (**sshd**, **inetd**, **cron**, ...) as Windows services under the SYSTEM (or LocalSystem in the GUI) account and everything just works. Unfortunately that's too simple. Using **NtCreateToken** has a few drawbacks.

First of all, beginning with Windows Server 2003, the permission "Create a token object" gets explicitly removed from the SYSTEM user's access token, when starting services under that account. That requires us to create a new account with this specific permission just to run this kind of services. But that's a minor problem.

A more important problem is that using **NtCreateToken** is not sufficient to create a new logon session for the new user. What does that mean? Every logon usually creates a new logon session. A logon session has a couple of attributes which are unique to the session. One of these attributes is the fact, that Windows functions identify the user domain and user name not by the SID of the access token owner, but only by the logon session the process is running under.

This has the following unfortunate consequence. Consider a service started under the SYSTEM account (up to Windows XP) switches the user context to DOMAIN\my_user using a token created directly by calling the **NtCreateToken** function. A process running under this new access token might want to know under which user account it's running. The corresponding SID is returned correctly, for instance S-1-5-21-1234-5678-9012-77777. However, if the same process asks the OS for the user name of this SID something wierd happens. For instance, the **LookupAccountSid** function will not return "DOMAIN\my_user", but "NT AUTHORITY\SYSTEM" as the user name.

You might ask "So what?" After all, this only **looks** bad, but functionality and permission-wise everything should be ok. And Cygwin knows about this shortcoming so it will return the correct Cygwin username when asked. Unfortunately this is more complicated. Some native, non-Cygwin Windows applications will misbehave badly in this situation. A well-known example are certain versions of Visual-C++.

Last but not least, you don't have the usual comfortable access to network shares. The reason is that the token has been created without knowing the password. The password are your credentials necessary for network access. Thus, if you logon with a password, the password is stored hidden as "token credentials" within the access token and used as default logon to access network resources. Since these credentials are missing from the token created with **NtCreateToken**, you only can access network shares from the new user's process tree by using explicit authentication, on the command line for instance:

```
bash$ net use '\\server\share' /user:DOMAIN\my_user my_users_password
```

Note that, on some systems, you can't even define a drive letter to access the share, and under some circumstances the drive letter you choose collides with a drive letter already used in another session. Therefore it's better to get used to accessing these shares using the UNC path as in

```
bash$ grep foo //server/share/foofile
```

2.5.8. Switching the user context without password, Method 2: LSA authentication package

Caveat: The method described in this chapter only works starting with Windows 2000. Windows NT4 users have to use one of the other methods described in this document.

We're looking for another way to switch the user context without having to provide the password. Another technique is to create an LSA authentication package. LSA is an acronym for "Local Security Authority" which is a protected part of the operating system which only allows changes to become active when rebooting the system after the change. Also, as soon as the LSA encounters serious problems (for instance, one of the protected LSA processes died), it triggers a system reboot. LSA is the part of the OS which cares for the user logons and which also creates logon sessions.

An LSA authentication package is a DLL which has to be installed as part of the LSA. This is done by tweaking a special registry key. Cygwin provides such an authentication package. It has to be installed and the machine has to be rebooted to activate it. This is the job of the shell script `/usr/bin/cyglsa-config` which is part of the Cygwin package.

After running `/usr/bin/cyglsa-config` and rebooting the system, the LSA authentication package is used by Cygwin when **set(e)uid** is called by an application. The created access token using this method has its own logon session.

This method has two advantages over the **NtCreateToken** method.

The very special and very dangerous "Create a token object" user right is not required by a user using this method. Other privileged user rights are still necessary, especially the "Act as part of the operating system" right, but that's just business as usual.

The user is correctly identified, even by delicate native applications which choke on that using the **NtCreateToken** method.

Disadvantages? Yes, sure, this is Windows. The access token created using LSA authentication still lacks the credentials for network access. After all, there still hasn't been any password authentication involved. The requirement to reboot after every installation or deinstallation of the cygwin LSA authentication DLL is just a minor inconvenience compared to that...

Nevertheless, this is already a lot better than what we get by using **NtCreateToken**, isn't it?

2.5.9. Switching the user context without password, Method

3: With password

Ok, so we have solved almost any problem, except for the network access problem. Not being able to access network shares without having to specify a cleartext password on the command line or in a script is a harsh problem for automated logons for testing purposes and similar stuff.

Fortunately there is a solution, but it has its own drawbacks. But, first things first, how does it work? The title of this section says it all. Instead of trying to logon without password, we just logon with password. The password gets stored two-way encrypted in a hidden, obfuscated area of the registry, the LSA private registry area. This part of the registry contains, for instance, the passwords of the Windows services which run under some non-default user account.

So what we do is to utilize this registry area for the purpose of **set(e)uid**. The Cygwin command **passwd -R** allows a user to specify his/her password for storage in this registry area. When this user tries to login using ssh with public key authentication, Cygwin's **set(e)uid** examines the LSA private registry area and searches for a Cygwin specific key which contains the password. If it finds it, it calls **LogonUser** under the hood, using this password. If that works, **LogonUser** returns an access token with all credentials necessary for network access.

For good measure, and since this way to implement **set(e)uid** is not only used by Cygwin but also by Microsoft's SFU (Services for Unix), we also look for a key stored by SFU (using the SFU command **regpwd**) and use that if it's available.

We got it. A full access token with its own logon session, with all network credentials. Hmm, that's heaven...

Back on earth, what about the drawbacks?

First, adding a password to the LSA private registry area requires administrative access. So calling **passwd -R** as a normal user will fail! Cygwin provides a workaround for this. If **cygserver** is started as a service running under the SYSTEM account (which is the default way to run **cygserver**) you can use **passwd -R** as normal, non-privileged user as well.

Second, as aforementioned, the password is two-way encrypted in a hidden, obfuscated registry area. Only SYSTEM has access to this area for listing purposes, so, even as an administrator, you can't examine this area with regedit. Right? No. Every administrator can start regedit as SYSTEM user:

```
bash$ date
Tue Dec  2 16:28:03 CET 2008
bash$ at 16:29 /interactive regedit.exe
```

Additionally, if an administrator knows under which name the private key is stored (which is well-known since the algorithms used to create the Cygwin and SFU keys are no secret), every administrator can access the password of all keys stored this way in the registry.

Conclusion: If your system is used exclusively by you, and if you're also the only administrator of your system, and if your system is adequately locked down to prevent malicious access, you can safely use this method. If your machine is part of a network which has dedicated administrators, and you're not one of these administrators, but you (think you) can trust your administrators, you can probably safely use this method.

In all other cases, don't use this method. You have been warned.

2.5.10. Switching the user context, how does it all fit together?

Now we learned about four different ways to switch the user context using the **set(e)uid** system call, but how does **set(e)uid** really work? Which method does it use now?

The answer is, all four of them. So here's a brief overview what **set(e)uid** does under the hood:

- When **set(e)uid** is called, it tests if the user context had been switched by an earlier call already, and if the new user account is the privileged user account under which the process had been started originally. If so, it just switches to the original access token of the process it had been started with.
- Next, it tests if an access token has been stored by an earlier call to **cygwin_set_impersonation_token**. If so, it tests if that token matches the requested user account. If so, the stored token is used for the user context switch.

  If not, there's no predefined token which can just be used for the user context switch, so we have to create a new token. The order is as follows.

- Check if the user has stored the logon password in the LSA private registry area, either under a Cygwin key, or under a SFU key. If so, use this to call **LogonUser**. If this succeeds, we use the resulting token for the user context switch.
- Otherwise, check if the Cygwin-specifc LSA authentication package has been installed and is functional. If so, use the appropriate LSA calls to communicate with the Cygwin LSA authentication package and use the returned token.
- Last chance, try to use the **NtCreateToken** call to create a token. If that works, use this token.
- If all of the above fails, our process has insufficient privileges to switch the user context at all, so **set(e)uid** fails and returns -1, setting errno to EPERM.

2.6. Customizing bash

To set up bash so that cut and paste work properly, click on the "Properties" button of the window, then on the "Misc" tab. Make sure that "QuickEdit mode" and "Insert mode" are checked. These settings will be remembered next time you run bash from that shortcut. Similarly you can set the working directory inside the "Program" tab. The entry "%HOME%" is valid, but requires that you set HOME in the Windows environment.

Your home directory should contain three initialization files that control the behavior of bash. They are `.profile`, `.bashrc` and `.inputrc`. The Cygwin base installation creates stub files when you start bash for the first time.

`.profile` (other names are also valid, see the bash man page) contains bash commands. It is executed when bash is started as login shell, e.g. from the command **bash --login**. This is a useful place to define and export environment variables and bash functions that will be used by bash and the programs invoked by bash. It is a good place to redefine PATH if needed. We recommend adding a ":." to the end of PATH to also search the current working directory (contrary to DOS, the local directory is not searched by default). Also to avoid delays you should either **unset** MAILCHECK or define MAILPATH to point to your existing mail inbox.

`.bashrc` is similar to `.profile` but is executed each time an interactive bash shell is launched. It serves to define elements that are not inherited through the environment, such as aliases. If you do not use login shells, you may want to put the contents of `.profile` as discussed above in this file instead.

```
shopt -s nocaseglob
```

will allow bash to glob filenames in a case-insensitive manner. Note that `.bashrc` is not called automatically for login shells. You can source it from `.profile`.

`.inputrc` controls how programs using the readline library (including **bash**) behave. It is loaded automatically. For full details see the `Function and Variable Index` section of the GNU readline manual. Consider the following settings:

```
# Ignore case while completing
set completion-ignore-case on
# Make Bash 8bit clean
set meta-flag on
set convert-meta off
set output-meta on
```

The first command makes filename completion case insensitive, which can be convenient in a Windows environment. The next three commands allow **bash** to display 8-bit characters, useful for languages with accented characters. Note that tools that do not use readline for display, such as **less** and **ls**, require additional settings, which could be put in your `.bashrc`:

```
alias less='/bin/less -r'
alias ls='/bin/ls -F --color=tty --show-control-chars'
```

Chapter 3. Using Cygwin

This chapter explains some key differences between the Cygwin environment and traditional UNIX systems. It assumes a working knowledge of standard UNIX commands.

3.1. Mapping path names

3.1.1. Introduction

Cygwin supports both Win32- and POSIX-style paths. Directory delimiters may be either forward slashes or backslashes. Paths using backslashes are always handled as Win32 paths. POSIX paths must only use forward slashes as delimiter, otherwise they are treated as Win32 paths and file access might fail in surprising ways. Note that the usage of Win32 paths, though possible, is deprecated, since it circumvents important internal path handling mechanisms. UNC pathnames (starting with two slashes and a network name) are also supported.

POSIX operating systems (such as Linux) do not have the concept of drive letters. Instead, all absolute paths begin with a slash (instead of a drive letter such as "c:") and all file systems appear as subdirectories (for example, you might buy a new disk and make it be the `/disk2` directory).

Because many programs written to run on UNIX systems assume the existance of a single unified POSIX file system structure, Cygwin maintains a special internal POSIX view of the Win32 file system that allows these programs to successfully run under Windows. Cygwin uses this mapping to translate from POSIX to Win32 paths as necessary.

3.1.2. The Cygwin Mount Table

The `/etc/fstab` file is used to map Win32 drives and network shares into Cygwin's internal POSIX directory tree. This is a similar concept to the typical UNIX fstab file. The mount points stored in `/etc/fstab` are globally set for all users. Sometimes there's a requirement to have user specific mount points. The Cygwin DLL supports user specific fstab files. These are stored in the directory `/etc/fstab.d` and the name of the file is the Cygwin username of the user, as it's stored in the `/etc/passwd` file. The content of the user specifc file is identical to the system-wide `fstab` file.

The file fstab contains descriptive information about the various file systems. fstab is only read by programs, and not written; it is the duty of the system administrator to properly create and maintain this file. Each filesystem is described on a separate line; fields on each line are separated by tabs or spaces. Lines starting with '#' are comments.

The first field describes the block special device or remote filesystem to be mounted. On Cygwin, this is the native Windows path which the mount point links in. As path separator you MUST use a slash. Usage of a backslash might lead to unexpected results. UNC paths (using slashes, not backslashes) are allowed. If the path contains spaces these can be escaped as `'\040'`.

The second field describes the mount point for the filesystem. If the name of the mount point contains spaces these can be escaped as '\040'.

The third field describes the type of the filesystem. Cygwin supports any string here, since the file system type is usually not evaluated. The notable exception is the file system type cygdrive. This type is used to set the cygdrive prefix.

The fourth field describes the mount options associated with the filesystem. It is formatted as a comma separated list of options. It contains at least the type of mount (binary or text) plus any additional options appropriate to the filesystem type. Recognized options are binary, text, nouser, user, exec, notexec, cygexec, nosuid, posix=[0|1]. The meaning of the options is as follows.

```
acl       - Cygwin uses the filesystem's access control lists (ACLs) to
            implement real POSIX permissions (default).  This flag only
     affects filesystems supporting ACLs (NTFS) and is ignored
     otherwise.
auto      - Ignored.
binary    - Files default to binary mode (default).
cygexec   - Treat all files below mount point as cygwin executables.
exec      - Treat all files below mount point as executable.
noacl     - Cygwin ignores filesystem ACLs and only fakes a subset of
     permission bits based on the DOS readonly attribute.  This
     behaviour is the default on FAT and FAT32.  The flag is
     ignored on NFS filesystems.
nosuid    - No suid files are allowed (currently unimplemented).
notexec   - Treat all files below mount point as not executable.
nouser    - Mount is a system-wide mount.
override  - Force the override of an immutable mount point (currently "/").
posix=0   - Switch off case sensitivity for paths under this mount point.
posix=1   - Switch on case sensitivity for paths under this mount point
     (default).
text      - Files default to CRLF text mode line endings.
user      - Mount is a user mount.
```

While normally the execute permission bits are used to evaluate executability, this is not possible on filesystems which don't support permissions at all (like FAT/FAT32), or if ACLs are ignored on filesystems supporting them (see the aforementioned `acl` mount option). In these cases, the following heuristic is used to evaluate if a file is executable: Files ending in certain extensions (.exe, .com, .bat, .btm, .cmd) are assumed to be executable. Files whose first two characters begin with '#!' are also considered to be executable. The `exec` option is used to instruct Cygwin that the mounted file is "executable". If the `exec` option is used with a directory then all files in the directory are executable. This option allows other files to be marked as executable and avoids the overhead of opening each file to check for a '#!'. The `cygexec` option is very similar to `exec`, but also prevents Cygwin from setting up commands and environment variables for a normal Windows program, adding another small performance gain. The opposite of these options is the `notexec` option, which means that no files should be marked as executable under that mount point.

A correct root directory is quite essential to the operation of Cygwin. A default root directory is evaluated at startup so a `fstab` entry for the root directory is not necessary. If it's wrong, nothing will work as expected. Therefore, the root directory evaluated by Cygwin itself is treated as an immutable mount point and can't be overridden in /etc/fstab... unless you think you really know what you're doing. In this case, use the `override` flag in the options field in the `/etc/fstab` file. Since this is a dangerous thing to do, do so at your own risk.

`/usr/bin` and `/usr/lib` are by default also automatic mount points generated by the Cygwin DLL similar to the way the root directory is evaluated. `/usr/bin` points to the directory the Cygwin DLL is installed in, `/usr/lib` is supposed to point to the `/lib` directory. This choice is safe and usually shouldn't be changed. An fstab entry for them is not required.

`nouser` mount points are not overridable by a later call to **mount**. Mount points given in `/etc/fstab` are by default `nouser` mount points, unless you specify the option `user`. This allows the administrator to set certain paths so that they are not overridable by users. In contrast, all mount points in the user specific fstab file are `user` mount points.

The fifth and sixth field are ignored. They are so far only specified to keep a Linux-like fstab file layout.

Note that you don't have to specify an fstab entry for the root dir, unless you want to have the root dir pointing to somewhere entirely different (hopefully you know what you're doing), or if you want to mount the root dir with special options (for instance, as text mount).

Example entries:

- Just a normal mount point:

  ```
  c:/foo /bar fat32 binary 0 0
  ```
- A mount point for a textmode mount with case sensitivity switched off:

  ```
  C:/foo /bar/baz ntfs text,posix=0 0 0
  ```
- A mount point for a Windows directory with spaces in it:

  ```
  C:/Documents\040and\040Settings /docs ext3 binary 0 0
  ```
- A mount point for a remote directory without ACL support:

  ```
  //server/share/subdir /srv/subdir smbfs binary,noacl 0 0
  ```
- This is just a comment:

  ```
  # This is just a comment
  ```
- Set the cygdrive prefix to /mnt:

  ```
  none /mnt cygdrive binary 0 0
  ```

Whenever Cygwin generates a Win32 path from a POSIX one, it uses the longest matching prefix in the mount table. Thus, if `C:` is mounted as `/c` and also as `/`, then Cygwin would translate `C:/foo/bar` to `/c/foo/bar`. This translation is normally only used when trying to derive the POSIX equivalent current directory. Otherwise, the handling of MS-DOS filenames bypasses the mount table.

If you want to see the current set of mount points valid in your session, you can invoking the Cygwin tool **mount** without arguments:

Example 3-1. Displaying the current set of mount points

```
bash$ mount
f:/cygwin/bin on /usr/bin type system (binary,auto)
f:/cygwin/lib on /usr/lib type system (binary,auto)
f:/cygwin on / type system (binary,auto)
e:/src on /usr/src type system (binary)
c: on /cygdrive/c type user (binary,posix=0,user,noumount,auto)
e: on /cygdrive/e type user (binary,posix=0,user,noumount,auto)
```

You can also use the **mount** command to add new mount points, and the **umount** to delete them. However, since they are only noted in memory, these mount points will disappear as soon as your last Cygwin process ends. See Section 3.7.8 and Section 3.7.15 for more information.

> **Note:** When you upgrade an existing older Cygwin installation to Cygwin 1.7, your old system mount points (stored in the HKEY_LOCAL_MACHINE branch of your registry) are read by a script and the `/etc/fstab` file is generated from these entries. Note that entries for `/`, `/usr/bin`, and `/usr/lib` are *never* generated.
>
> The old user mount points in your HKEY_CURRENT_USER branch of the registry are not used to generate `/etc/fstab`. If you want to create a user specific `/etc/fstab.d/${USER}` file from your old entries, there's a script available which does exactly that for you, `bin/copy-user-registry-fstab`. Just start the script and it will create your user specific fstab file. Stop all your Cygwin processes and restart them, and you can simply use your old user mount points as before.

3.1.3. The cygdrive path prefix

As already outlined in Section 1.6.3, you can access arbitary drives on your system by using the cygdrive path prefix. The default value for this prefix is `/cygdrive`, and a path to any drive can be constructed by using the cygdrive prefix and appending the drive letter as subdirectory, like this:

```
bash$ ls -l /cygdrive/f/somedir
```

This lists the content of the directory F:\somedir.

The cygdrive prefix is a virtual directory under which all drives on a system are subsumed. The mount options of the cygdrive prefix is used for all file access through the cygdrive prefixed drives. For instance, assuming the cygdrive mount options are `binary,posix=0`, then any file `/cygdrive/x/file` will be opened in binary mode by default (mount option `binary`, and the case of the filename doesn't matter (mount option `posix=0`.

The cygdrive prefix may be changed in the fstab file as outlined above. Please note that you must not use the cygdrive prefix for any other mount point. For instance this:

```
none /cygdrive cygdrive binary 0 0
D:   /cygdrive/d somefs text 0 0
```

will not make file access using the /mnt/d path prefix suddenly using textmode. If you want to mount any drive explicitly in another mode than the cygdrive prefix, use a distinct path prefix:

```
none /cygdrive cygdrive binary 0 0
D:   /mnt/d somefs text 0 0
```

3.1.4. Additional Path-related Information

The **cygpath** program provides the ability to translate between Win32 and POSIX pathnames in shell scripts. See Section 3.7.2 for the details.

The HOME, PATH, and LD_LIBRARY_PATH environment variables are automatically converted from Win32 format to POSIX format (e.g. from `c:/cygwin\bin` to `/bin`, if there was a mount from that Win32 path to that POSIX path) when a Cygwin process first starts.

Symbolic links can also be used to map Win32 pathnames to POSIX. For example, the command **ln -s //pollux/home/joe/data /data** would have about the same effect as creating a mount point from `//pollux/home/joe/data` to `/data` using **mount**, except that symbolic links cannot set the default file access mode. Other differences are that the mapping is distributed throughout the file system and proceeds by iteratively walking the directory tree instead of matching the longest prefix in a kernel table. Note that symbolic links will only work on network drives that are properly configured to support the "system" file attribute. Many do not do so by default (the Unix Samba server does not by default, for example).

3.2. Text and Binary modes

3.2.1. The Issue

On a UNIX system, when an application reads from a file it gets exactly what's in the file on disk and the converse is true for writing. The situation is different in the DOS/Windows world where a file can be opened in one of two modes, binary or text. In the binary mode the system behaves exactly as in UNIX. However on writing in text mode, a NL (\n, ^J) is transformed into the sequence CR (\r, ^M) NL.

This can wreak havoc with the seek/fseek calls since the number of bytes actually in the file may differ from that seen by the application.

The mode can be specified explicitly as explained in the Programming section below. In an ideal DOS/Windows world, all programs using lines as records (such as **bash**, **make**, **sed** ...) would open files (and change the mode of their standard input and output) as text. All other programs (such as **cat**, **cmp**, **tr** ...) would use binary mode. In practice with Cygwin, programs that deal explicitly with object files specify binary mode (this is the case of **od**, which is helpful to diagnose CR problems). Most other programs (such as **cat**, **cmp**, **tr**) use the default mode.

3.2.2. The default Cygwin behavior

The Cygwin system gives us some flexibility in deciding how files are to be opened when the mode is not specified explicitly. The rules are evolving, this section gives the design goals.

a. If the filename is specified as a POSIX path and it appears to reside on a file system that is mounted (i.e. if its pathname starts with a directory displayed by **mount**), then the default is specified by the mount flag. If the file is a symbolic link, the mode of the target file system applies.
b. If the file is specified via a MS-DOS pathname (i.e., it contains a backslash or a colon), the default is binary.
c. Pipes, sockets and non-file devices are opened in binary mode. For pipes opened through the pipe() system call you can use the setmode() function (see Section 3.2.5 to switch to textmode. For pipes opened through popen(), you can simply specify text or binary mode just like in calls to fopen().
d. Sockets and other non-file devices are always opened in binary mode.
e. When redirecting, the Cygwin shells uses rules (a-d). Non-Cygwin shells always pipe and redirect with binary mode. With non-Cygwin shells the commands **cat filename | program** and **program < filename** are not equivalent when `filename` is on a text-mounted partition.

3.2.3. Example

To illustrate the various rules, we provide scripts to delete CRs from files by using the **tr** program, which can only write to standard output. The script

```
#!/bin/sh
# Remove \r from the file given as argument
tr -d '\r' < "$1" > "$1".nocr
```

will not work on a text mounted systems because the \r will be reintroduced on writing. However scripts such as

```
#!/bin/sh
# Remove \r from the file given as argument
tr -d '\r' | gzip | gunzip > "$1".nocr
```

and the .bat file

```
REM Remove \r from the file given as argument
@echo off
tr -d \r < %1 > %1.nocr
```

work fine. In the first case we rely on **gunzip** to set its output to binary mode, possibly overriding the mode used by the shell. In the second case we rely on the DOS shell to redirect in binary mode.

3.2.4. Binary or text?

UNIX programs that have been written for maximum portability will know the difference between text and binary files and act appropriately under Cygwin. Most programs included in the official Cygwin distributions should work well in the default mode.

Binmode is the best choice usually since it's faster and easier to handle, unless you want to exchange files with native Win32 applications. It makes most sense to keep the Cygwin distribution and your Cygwin home directory in binmode and generate text files in binmode (with UNIX LF lineendings). Most Windows applications can handle binmode files just fine. A notable exception is the mini-editor **Notepad**, which handles UNIX lineendings incorrectly and only produces output files with DOS CRLF lineendings.

You can convert files between CRLF and LF lineendings by using certain tools in the Cygwin distribution like **d2u** and **u2d** from the cygutils package. You can also specify a directory in the mount table to be mounted in textmode so you can use that directory for exchange purposes.

As application programmer you can decide on a file by file base, or you can specify default open modes depending on the purpose for which the application open files. See the next section for a description of your choices.

3.2.5. Programming

In the `open()` function call, binary mode can be specified with the flag `O_BINARY` and text mode with `O_TEXT`. These symbols are defined in `fcntl.h`.

In the `fopen()` and `popen()` function calls, binary mode can be specified by adding a `b` to the mode string. Text mode is specified by adding a `t` to the mode string.

The mode of a file can be changed by the call `setmode(fd,mode)` where `fd` is a file descriptor (an integer) and `mode` is `O_BINARY` or `O_TEXT`. The function returns `O_BINARY` or `O_TEXT` depending on the mode before the call, and `EOF` on error.

There's also a convenient way to set the default open modes used in an application by just linking against various object files provided by Cygwin. For instance, if you want to make sure that all files are always opened in binary mode by an application, regardless of the mode of the underlying mount point, just add the file `/lib/binmode.o` to the link stage of the application in your project, like this:

```
$ gcc my_tiny_app.c /lib/binmode.o -o my_tiny_app
```

This adds code which sets the default open mode for all files opened by **my_tiny_app** to binary for reading and writing.

Cygwin provides the following object files to set the default open mode just by linking an application against them:

- ```
  /lib/automode.o      -  Open files for reading in textmode
                         Open files for writing in binary mode
  ```
- ```
  /lib/binmode.o       -  Open files for reading and writing in binary mode
  ```
- ```
  /lib/textmode.o      -  Open files for reading and writing in textmode
  ```
- ```
  /lib/textreadmode.o  -  Open files for reading in textmode
                         Keep default behaviour for writing.
  ```

3.3. File permissions

On FAT or FAT32 filesystems, files are always readable, and Cygwin uses the DOS read-only attribute to determine if they are writable. Files are considered to be executable if the filename ends with .bat, .com or .exe, or if its content starts with #!. Consequently **chmod** can only affect the "w" mode, it silently ignores actions involving the other modes. This means that **ls -l** needs to open and read files. It can thus be relatively slow.

On NTFS, file permissions are evaluated using the Access Control Lists (ACLs) attached to a file. This can be switched off by using the "noacl" option to the respective mount point in the `/etc/fstab` or `/etc/fstab.d/$USER` file. For more information on file permissions, see Section 2.5.

On NFS shares, file permissions are exactly the POSIX permissions transmitted from the server using the NFSv3 protocol, if the NFS client is the one from Microsoft's "Services For Unix", or the one built into Windows Vista or later.

Only the user and group ownership is not necessarily correct.

3.4. Special filenames

3.4.1. Special files in /etc

Certain files in Cygwin's `/etc` directory are read by Cygwin before the mount table has been established. The list of files is

```
/etc/fstab
/etc/fstab.d/$USER
/etc/passwd
/etc/group
```

These file are read using native Windows NT functions which have no notion of Cygwin symlinks or POSIX paths. For that reason there are a few requirements as far as `/etc` is concerned.

To access these files, the Cygwin DLL evaluates it's own full Windows path, strips off the innermost directory component and adds "\etc". Let's assume the Cygwin DLL is installed as `C:\cygwin\bin\cygwin1.dll`. First the DLL name as well as the innermost directory (`bin`) is stripped off: `C:\cygwin\`. Then "etc" and the filename to look for is attached: `C:\cygwin\etc\fstab`. So the /etc directory must be parallel to the directory in which the cygwin1.dll exists and `/etc` must not be a Cygwin symlink pointing to another directory. Consequentially none of the files from the above list, including the directory `/etc/fstab.d` is allowed to be a Cygwin symlink either.

However, native NTFS symlinks and reparse points are transparent when accessing the above files so all these files as well as `/etc` itself may be NTFS symlinks or reparse points.

Last but not least, make sure that these files are world-readable. Every process of any user account has to read these files potentially, so world-readability is essential. The only exception are the user specific files `/etc/fstab.d/$USER`, which only have to be readable by the $USER user account itself.

3.4.2. Invalid filenames

Filenames invalid under Win32 are not necessarily invalid under Cygwin since release 1.7.0. There are a few rules which apply to Windows filenames. Most notably, DOS device names like `AUX`, `COM1`, `LPT1` or `PRN` (to name a few) cannot be used as filename or extension in a native Win32 application. So filenames like `prn.txt` or `foo.aux` are invalid filenames for native Win32 applications.

This restriction doesn't apply to Cygwin applications. Cygwin can create and access files with such names just fine. Just don't try to use these files with native Win32 applications.

3.4.3. Forbidden characters in filenames

Win32 filenames can't contain trailing dots and spaces for backward compatibility. When trying to create files with trailing dots or spaces, all of them are removed before the file is created. This restriction only affects native Win32 applications. Cygwin applications can create and access files with trailing dots and spaces without problems.

Some characters are disallowed in filenames on Windows filesystems:

```
"    *    :    <    >    ?    |    \
```

Cygwin can't fix this, but it has a method to workaround this restriction. All of the above characters, except for the backslash, are converted to special UNICODE characters in the range 0xf000 to 0xf0ff (the "Private use area") when creating or accessing files.

3.4.4. Filenames with unusual (foreign) characters

Windows filesystems use the Unicode character set in the UTF-16 encoding to store filename information. If you don't use the UTF-8 character set (see Section 2.4) then there's a chance that a filename is using one or more characters which have no representation in the character set you're using.

For instance, there are no Chinese characters in the ISO-8859-1 character set. So, converting a filename containing a Chinese character to ISO-8859-1 leaves you with a wrongly converted filename, for instance, containing a question mark '?' as replacement for the unconvertable character. When trying to access the file, Cygwin has to convert the filename back to UTF-16. However, this doesn't result in the original filename because the question mark will not translate back to the original Chinese character, but to a simple question mark instead. This in turn results in strange "File not found" messages.

> **Note:** In the default "C" locale, Cygwin creates filenames using the UTF-8 charset. This will always result in some valid filename by default, but again might impose problems when switching to a non-"C" or non-"UTF-8" charset.

Note: To avoid this scenario altogether, always use UTF-8 as the character set.

If you don't want or can't use UTF-8 as character set for whatever reason, you will nevertheless be able to access the file. How does that work? When Cygwin converts the filename from UTF-16 to your character set, it recognizes characters which can't be converted. If that occurs, Cygwin replaces the non-convertible character with a special character sequence. The sequence starts with an ASCII CAN character (hex code 0x18, equivalent Control-X), followed by the UTF-8 representation of the character. The result is a filename containing some ugly looking characters. While it doesn't *look* nice, it *is* nice, because Cygwin knows how to convert this filename back to UTF-16. The filename will be converted using your usual character set. However, when Cygwin recognizes an ASCII CAN character, it skips over the ASCII CAN and handles the following bytes as a UTF-8 character. Thus, the filename is symmetrically converted back to UTF-16 and you can access the file.

Note: Please be aware that this method is not entirely foolproof. In some character set combinations it might not work for certain native characters.

Only by using the UTF-8 charset you can avoid this problem safely.

3.4.5. Case sensitive filenames

In the Win32 subsystem filenames are only case-preserved, but not case-sensitive. You can't access two files in the same directory which only differ by case, like `Abc` and `aBc`. While NTFS (and some remote filesystems) support case-sensitivity, the NT kernel starting with Windows XP does not support it by default. Rather, you have to tweak a registry setting and reboot. For that reason, case-sensitivity can not be supported by Cygwin, unless you change that registry value.

If you really want case-sensitivity in Cygwin, you can switch it on by setting the registry value

```
HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\kernel\obcaseinsensitive
```

to 0 and reboot the machine. For least surprise, Cygwin expects this registry value also on Windows NT4 and Windows 2000, which usually both don't know this registry key. If you want case-sensitivity on these systems, create that registry value and set it to 0. On these systems (and **only** on these systems) you don't have to reboot to bring it into effect, rather stopping all Cygwin processes and then restarting them is sufficient.

Note: When installing Microsoft's Services For Unix (SFU), you're asked if you want to use case-sensitive filenames. If you answer "yes" at this point, the installer will change the aforementioned registry value to 0, too. So, if you have SFU installed, there's some chance that the registry value is already set to case sensitivity.

After you set this registry value to 0, Cygwin will be case-sensitive by default on NTFS and NFS filesystems. Be aware that using two filenames which only differ by case might result in some weird interoperability issues with native Win32 applications. You're using case-sensitivity at your own risk. You have been warned!

Even if you use case-sensitivity, it might be feasible to switch to case-insensitivity for certain paths for better interoperability with native Win32 applications (even if it's just Windows Explorer). You can do this on a per-mount point base, by using the "posix=0" mount option in `/etc/fstab`, or your `/etc/fstab.d/$USER` file.

`/cygdrive` paths are case-insensitive by default. The reason is that the native Windows %PATH% environment variable is not always using the correct case for all paths in it. As a result, if you use case-sensitivity on the `/cygdrive` prefix, your shell might claim that it can't find Windows commands like **attrib** or **net**. To ease the pain, the `/cygdrive` path is case-insensitive by default and you have to use the "posix=1" setting explicitly in `/etc/fstab` or `/etc/fstab.d/$USER` to switch it to case-sensitivity, or you have to make sure that the native Win32 %PATH% environment variable is using the correct case for all paths throughout.

Note that mount points as well as device names and virtual paths like /proc are always case-sensitive! The only exception are the subdirectories and filenames under /proc/registry, /proc/registry32 and /proc/registry64. Registry access is always case-insensitive. Read on for more information.

3.4.6. POSIX devices

There is no need to create a POSIX `/dev` directory as Cygwin automatically simulates it internally. These devices cannot be seen with the command **ls /dev/** although commands such as **ls /dev/tty** work fine. If you want to be able to see all well-known devices in `/dev/`, you can use Igor Pechtchanski's create_devices.sh (http://cygwin.com/ml/cygwin/2004-03/txt00028.txt) script. This script does not add the raw disk devices, though. Again, it's not necessary to see an existing device in /dev to access it. The script is just for the fun of it.

Cygwin supports the following character devices commonly found on POSIX systems:

```
/dev/null
/dev/zero
/dev/full

/dev/console Pseudo device name for the standard console window created
  by Windows.  Same as the one used for cmd.exe.  Every one
  of them has this name.  It's not quite comparable with the
  console device on UNIX machines.

/dev/tty The current tty of a session running in a pseudo tty.
/dev/ptmx Pseudo tty master device.
/dev/ttym
```

```
/dev/tty0 Pseudo ttys are numbered from /dev/tty0 upwards as they are
/dev/tty1 requested.
...

/dev/ttyS0 Serial communication devices.  ttyS0 == Win32 COM1,
/dev/ttyS1 ttyS1 == COM2, etc.
...

/dev/pipe
/dev/fifo

/dev/mem The physical memory of the machine.  Note that access to the
/dev/port physical memory has been restricted with Windows Server 2003.
/dev/kmem Since this OS, you can't access physical memory from user space.

/dev/kmsg Kernel message pipe, for usage with sys logger services.

/dev/random Random number generator.
/dev/urandom

/dev/dsp Default sound device of the system.
```

Cygwin also has several Windows-specific devices:

```
/dev/com1 The serial ports, starting with COM1 which is the same as ttyS0.
/dev/com2 Please use /dev/ttySx instead.
...

/dev/conin Same as Windows CONIN$.
/dev/conout Same as Windows CONOUT$.
/dev/clipboard The Windows clipboard, text only
/dev/windows The Windows message queue.
```

Block devices are accessible by Cygwin processes using fixed POSIX device names. These POSIX device names are generated using a direct conversion from the POSIX namespace to the internal NT namespace. E.g. the first harddisk is the NT internal device \device\harddisk0\partition0 or the first partition on the third harddisk is \device\harddisk2\partition1. The first floppy in the system is \device\floppy0, the first CD-ROM is \device\cdrom0 and the first tape drive is \device\tape0.

The mapping from physical device to the name of the device in the internal NT namespace can be found in various places. For hard disks and CD/DVD drives, the Windows "Disk Management" utility (part of the "Computer Management" console) shows that the mapping of "Disk 0" is \device\harddisk0. "CD-ROM 2" is \device\cdrom2. Another place to find this mapping is the "Device Management" console. Disks have a "Location" number, tapes have a "Tape Symbolic Name", etc. Unfortunately, the places where this information is found is not very well-defined.

For external disks (USB-drives, CF-cards in a cardreader, etc) you can use Cygwin to show the mapping. `/proc/partitions` contains a list of raw drives known to Cygwin. The **df** command shows a list of drives and their respective sizes. If you match the information between `/proc/partitions` and the **df** output, you should be able to figure out which external drive corresponds to which raw disk device name.

> **Note:** Apart from tape devices which are not block devices and are by default accessed directly, accessing mass storage devices raw is something you should only do if you know what you're doing and know how to handle the information. **Writing** to a raw mass storage device you should only do if you **really** know what you're doing and are aware of the fact that any mistake can destroy important information, for the device, and for you. So, please, handle this ability with care. **You have been warned.**

Last but not least, the mapping from POSIX /dev namespace to internal NT namespace is as follows:

```
POSIX device name     Internal NT device name

/dev/st0       \device\tape0, rewind
/dev/nst0       \device\tape0, no-rewind
/dev/st1       \device\tape1
/dev/nst1       \device\tape1
...
/dev/st15
/dev/nst15

/dev/fd0       \device\floppy0
/dev/fd1       \device\floppy1
...
/dev/fd15

/dev/sr0       \device\cdrom0
/dev/sr1       \device\cdrom1
...
/dev/sr15

/dev/scd0       \device\cdrom0
/dev/scd1       \device\cdrom1
...
/dev/scd15

/dev/sda       \device\harddisk0\partition0 (whole disk)
/dev/sda1       \device\harddisk0\partition1 (first partition)
...
/dev/sda15       \device\harddisk0\partition15 (fifteenth partition)

/dev/sdb       \device\harddisk1\partition0
/dev/sdb1       \device\harddisk1\partition1

[up to]

/dev/sddx       \device\harddisk127\partition0
/dev/sddx1       \device\harddisk127\partition1
...
```

```
/dev/sddx15       \device\harddisk127\partition15
```

if you don't like these device names, feel free to create symbolic links as they are created on Linux systems for convenience:

```
ln -s /dev/sr0 /dev/cdrom
ln -s /dev/nst0 /dev/tape
...
```

3.4.7. The .exe extension

Win32 executable filenames end with `.exe` but the `.exe` need not be included in the command, so that traditional UNIX names can be used. However, for programs that end in `.bat` and `.com`, you cannot omit the extension.

As a side effect, the **ls filename** gives information about `filename.exe` if `filename.exe` exists and `filename` does not. In the same situation the function call `stat("filename",..)` gives information about `filename.exe`. The two files can be distinguished by examining their inodes, as demonstrated below.

```
bash$ ls *
a      a.exe     b.exe
bash$ ls -i a a.exe
445885548 a       435996602 a.exe
bash$ ls -i b b.exe
432961010 b       432961010 b.exe
```

If a shell script `myprog` and a program `myprog.exe` coexist in a directory, the shell script has precedence and is selected for execution of **myprog**. Note that this was quite the reverse up to Cygwin 1.5.19. It has been changed for consistency with the rest of Cygwin.

The **gcc** compiler produces an executable named `filename.exe` when asked to produce `filename`. This allows many makefiles written for UNIX systems to work well under Cygwin.

3.4.8. The /proc filesystem

Cygwin, like Linux and other similar operating systems, supports the `/proc` virtual filesystem. The files in this directory are representations of various aspects of your system, for example the command **`cat /proc/cpuinfo`** displays information such as what model and speed processor you have.

One unique aspect of the Cygwin `/proc` filesystem is `/proc/registry`, see next section.

The Cygwin /proc is not as complete as the one in Linux, but it provides significant capabilities. The procps package contains several utilities that use it.

3.4.9. The /proc/registry filesystem

The /proc/registry filesystem provides read-only access to the Windows registry. It displays each KEY as a directory and each VALUE as a file. As anytime you deal with the Windows registry, use caution since changes may result in an unstable or broken system. There are additionally subdirectories called /proc/registry32 and /proc/registry64. They are identical to /proc/registry on 32 bit host OSes. On 64 bit host OSes, /proc/registry32 opens the 32 bit processes view on the registry, while /proc/registry64 opens the 64 bit processes view.

Reserved characters ('/', '\', ':', and '%') or reserved names (. and ..) are converted by percent-encoding:

```
bash$ regtool list -v '\HKEY_LOCAL_MACHINE\SYSTEM\MountedDevices'
...
\DosDevices\C: (REG_BINARY) = cf a8 97 e8 00 08 fe f7
...
bash$ cd /proc/registry/HKEY_LOCAL_MACHINE/SYSTEM
bash$ ls -l MountedDevices
...
-r--r----- 1 Admin SYSTEM  12 Dec 10 11:20 %5CDosDevices%5CC%3A
...
bash$ od -t x1 MountedDevices/%5CDosDevices%5CC%3A
0000000 cf a8 97 e8 00 08 fe f7 01 00 00 00
```

The unnamed (default) value of a key can be accessed using the filename @.

If a registry key contains a subkey and a value with the same name foo, Cygwin displays the subkey as foo and the value as foo%val.

3.4.10. The @pathnames

To circumvent the limitations on shell line length in the native Windows command shells, Cygwin programs expand their arguments starting with "@" in a special way. If a file pathname exists, the argument @pathname expands recursively to the content of pathname. Double quotes can be used inside the file to delimit strings containing blank space. Embedded double quotes must be repeated. In the following example compare the behaviors of the bash built-in **echo** and of the program **/bin/echo**.

Example 3-2. Using @pathname

```
bash$ echo  'This   is   "a     long"  line' > mylist
bash$ echo @mylist
@mylist
bash$ cmd
```

```
c:\> c:\cygwin\bin\echo @mylist
This is a     long line
```

3.5. The CYGWIN environment variable

3.5.1. Implemented options

The CYGWIN environment variable is used to configure many global settings for the Cygwin runtime system. It contains the options listed below, separated by blank characters. Many options can be turned off by prefixing with `no`.

- (no)dosfilewarning - If set, Cygwin will warn the first time a user uses an "MS-DOS" style path name rather than a POSIX-style path name. Defaults to set.
- (no)envcache - If set, environment variable conversions (between Win32 and POSIX) are cached. Note that this may cause problems if the mount table changes, as the cache is not invalidated and may contain values that depend on the previous mount table contents. Defaults to set.
- (no)export - If set, the final values of these settings are re-exported to the environment as CYGWIN again. Defaults to off.
- error_start:Win32filepath - if set, runs `Win32filepath` when cygwin encounters a fatal error, which is useful for debugging. `Win32filepath` is usually set to the path to **gdb** or **dumper**, for example `C:\cygwin\bin\gdb.exe`. There is no default set.
- forkchunk:32768 - causes `fork()` to copy memory some number of bytes at a time, in the above example 32768 bytes (32Kb) at a time. The default is to copy as many bytes as possible, which is preferable in most cases but may slow some older systems down.
- proc_retry:n - causes `fork()` and `exec*()` to retry n times when a child process fails due to certain windows-specific errors. These errors usually occur when processes are being started while a user is logging off.
- (no)glob[:ignorecase] - if set, command line arguments containing UNIX-style file wildcard characters (brackets, question mark, asterisk, escaped with \) are expanded into lists of files that match those wildcards. This is applicable only to programs running from a DOS command line prompt. Default is set.

  This option also accepts an optional `[no]ignorecase` modifer. If supplied, wildcard matching is case insensitive. The default is `noignorecase`

- (no)reset_com - if set, serial ports are reset to 9600-8-N-1 with no flow control when used. This is done at open time and when handles are inherited. Defaults to set.
- (no)strip_title - if set, strips the directory part off the window title, if any. Default is not set.
- (no)title - if set, the title bar reflects the name of the program currently running. Default is not set.

- (no)tty - if set, Cygwin enables extra support (i.e., termios) for UNIX-like ttys in the Windows console. It is not compatible with some Windows programs. Defaults to not set, in which case the tty is opened in text mode. Note that this has been changed such that ^D works as expected instead of ^Z, and is settable via **stty**. This option must be specified before starting a Cygwin shell and it cannot be changed in the shell. It should not be set when using other terminals (i.e., rxvt or xterm).
- (no)upcaseenv - if set, Cygwin converts all environment variables to all-uppercase, when a Cygwin process is started from a non-Cygwin native Windows process. This was the default behavior in releases prior to Cygwin 1.7. If not set, Cygwin does not change the case of environment variables, except for a restricted set to maintain minimal backward compatibility and for correct handling of certain essential variables. The current list of always uppercased variables is:

  ```
  ALLUSERSPROFILE
  COMMONPROGRAMFILES
  COMPUTERNAME
  COMSPEC
  HOME
  HOMEDRIVE
  HOMEPATH
  NUMBER_OF_PROCESSORS
  OS
  PATH
  PATHEXT
  PROCESSOR_ARCHITECTURE
  PROCESSOR_IDENTIFIER
  PROCESSOR_LEVEL
  PROCESSOR_REVISION
  PROGRAMFILES
  SYSTEMDRIVE
  SYSTEMROOT
  TEMP
  TERM
  TMP
  TMPDIR
  WINDIR
  ```

  Defaults to not set.

- (no)winsymlinks - if set, Cygwin creates symlinks as Windows shortcuts with a special header and the R/O attribute set. If not set, Cygwin creates symlinks as plain files with a magic number, a path and the system attribute set. Defaults to not set since plain file symlinks are faster to write and faster to read. The new style of plain file symlinks is incompatible with older Cygwin releases because the new symlinks use UTF-16 to encode the target filename, while the old symlinks used the current ANSI or OEM charset.

3.5.2. Obsolete options

Certain CYGWIN options available in past releases have been removed in Cygwin 1.7 for one reason or another. These obsolete options are listed below.

- (no)binmode - This option has been removed because all file opens default to binary mode, unless the open mode has been specified explicitly in the open(2) call.
- check_case - This option has been removed in favor of real case sensitivity and the per-mount option "posix=[0|1]". For more information, read the documentation in Section 3.1.2 and Section 3.4.5.
- codepage:[ansi|oem] - This option controled which character set is used for file and console operations. Since Cygwin is now doing all character conversion by itself, depending on the application call to the `setlocale()` function, and in turn by the setting of the environment variables $LANG, $LC_ALL, or $LC_CTYPE, this setting became superfluous.
- (no)ntea - This option has been removed since it only fakes security which is considered dangerous and useless. It also created an uncontrollably large file on FAT and was entirely useless on FAT32.
- (no)ntsec - This option has been removed in favor of the per-mount option "acl"/"noacl". For more information, read the documentation in Section 3.1.2.
- (no)server - Originally this option had to be enabled on the client side to use features only available when running **cygserver**. This option has been removed because Cygwin now always tries to contact cygserver if a function is called which requires cygserver being available. For more information, read the documentation in Section 3.6.
- (no)smbntsec - This option has been removed in favor of the per-mount option "acl"/"noacl". For more information, read the documentation in Section 3.1.2.
- (no)transparent_exe - This option has been removed because the behaviour it switched on is now the standard behaviour in Cygwin.
- (no)traverse - This option has been removed because traverse checking is not quite correctly implemented by Microsoft and it's behaviour has been getting worse with each new OS version. This complicates its usage so the option has been removed for now.

3.6. Cygserver

3.6.1. What is Cygserver?

Cygserver is a program which is designed to run as a background service. It provides Cygwin applications with services which require security arbitration or which need to persist while no other cygwin application is running.

The implemented services so far are:

- Control slave tty/pty handle dispersal from tty owner to other processes without compromising the owner processes' security.
- XSI IPC Message Queues.
- XSI IPC Semaphores.
- XSI IPC Shared Memory.
- Allows non-privileged users to store obfuscated passwords in the registry to be used by **setuid** and **seteuid** calls to create user tokens with network credentials. This service is used by **passwd -R**. Using the stored passwords in **set(e)uid** does not require running Cygserver. For details, see Section 2.5.5.

3.6.2. Cygserver command line options

Options to Cygserver take the normal UNIX-style '-X' or '--longoption' form. Nearly all options have a counterpart in the configuration file (see below) so setting them on the command line isn't really necessary. Command line options override settings from the Cygserver configuration file.

The one-character options are prepended by a single dash, the long variants are prepended with two dashes. Arguments to options are marked in angle brackets below. These are not part of the actual syntax but are used only to denote the arguments. Note that all arguments are required. Cygserver has no options with optional arguments.

The recognized options are:

- `-f, --config-file <file>`

  Use <file> as configuration file instead of the default configuration line. The default configuration file is /etc/cygserver.conf. The --help and --version options will print the default configuration pathname.

  This option has no counterpart in the configuration file, for obvious reasons.

- `-c, --cleanup-threads <num>`

  Number of threads started to perform cleanup tasks. Default is 2. Configuration file option: kern.srv.cleanup_threads

- `-r, --request-threads <num>`

  Number of threads started to serve application requests. Default is 10. The -c and -r options can be used to play with Cygserver's performance under heavy load conditions or on slow machines. Configuration file option: kern.srv.request_threads

- `-d, --debug`

Log debug messages to stderr. These will clutter your stderr output with a lot of information, typically only useful to developers.

- `-e, --stderr`

  Force logging to stderr. This is the default if stderr is connected to a tty. Otherwise, the default is logging to the system log. By using the -e, -E, -y, -Y options (or the appropriate settings in the configuration file), you can explicitly set the logging output as you like, even to both, stderr and syslog. Configuration file option: kern.log.stderr

- `-E, --no-stderr`

  Don't log to stderr. Configuration file option: kern.log.stderr

- `-y, --syslog`

  Force logging to the system log. This is the default, if stderr is not connected to a tty, e. g. redirected to a file. Configuration file option: kern.log.syslog

- `-Y, --no-syslog`

  Don't log to syslog. Configuration file option: kern.log.syslog

- `-l, --log-level <level>`

  Set the verbosity level of the logging output. Valid values are between 1 and 7. The default level is 6, which is relatively chatty. If you set it to 1, you will get only messages which are printed under severe conditions, which will result in stopping Cygserver itself. Configuration file option: kern.log.level

- `-m, --no-sharedmem`

  Don't start XSI IPC Shared Memory support. If you don't need XSI IPC Shared Memory support, you can switch it off here. Configuration file option: kern.srv.sharedmem

- `-q, --no-msgqueues`

  Don't start XSI IPC Message Queues. Configuration file option: kern.srv.msgqueues

- `-s, --no-semaphores`

  Don't start XSI IPC Semaphores. Configuration file option: kern.srv.semaphores

- `-S, --shutdown`

Shutdown a running daemon and exit. Other methods are sending a SIGHUP to the Cygserver PID or, if running as service, calling ‘net stop cygserver’ or ‘cygrunsrv -E cygserver’.

- `-h, --help`

  Output usage information and exit.

- `-v, --version`

  Output version information and exit.

3.6.3. How to start Cygserver

Before you run Cygserver for the first time, you should run the /usr/bin/cygserver-config script once. It creates the default configuration file and, upon request, installs Cygserver as service. The script only performs a default install, with no further options given to Cygserver when running as service. Due to the wide configurability by changing the configuration file, that’s typically not necessary.

You should always run Cygserver as a service under LocalSystem account. This is the way it is installed for you by the /usr/bin/cygserver-config script.

3.6.4. The Cygserver configuration file

Cygserver has many options, which allow you to customize the server to your needs. Customization is accomplished by editing the configuration file, which is by default /etc/cygserver.conf. This file is only read once, at startup of Cygserver. There’s no option to re-read the file at runtime by, say, sending a signal to Cygserver.

The configuration file determines how Cygserver operates. There are options which set the number of threads running in parallel, options for setting how and what to log and options to set various maximum values for the IPC services.

The default configuration file delivered with Cygserver is installed to /etc/defaults/etc. The /usr/bin/cygserver-config script copies it to /etc, giving you the option to overwrite an already existing file or to leave it alone. Therefore, the /etc file is safe to be changed by you, since it will not be overwritten by a later update installation.

The default configuration file contains many comments which describe everything needed to understand the settings. A comment at the start of the file describes the syntax rules for the file. The default options are shown in the file but are commented out.

It is generally a good idea to uncomment only options which you intend to change from the default values. Since reading the options file on Cygserver startup doesn't take much time, it's also considered good practice to keep all other comments in the file. This keeps you from searching for clues in other sources.

3.7. Cygwin Utilities

Cygwin comes with a number of command-line utilities that are used to manage the UNIX emulation portion of the Cygwin environment. While many of these reflect their UNIX counterparts, each was written specifically for Cygwin. You may use the long or short option names interchangeably; for example, `--help` and `-h` function identically. All of the Cygwin command-line utilities support the `--help` and `--version` options.

3.7.1. cygcheck

```
Usage: cygcheck PROGRAM [ -v ] [ -h ]
       cygcheck -c [ PACKAGE ... ] [ -d ]
       cygcheck -s [ -r ] [ -v ] [ -h ]
       cygcheck -k
       cygcheck -f FILE [ FILE ... ]
       cygcheck -l [ PACKAGE ... ]
       cygcheck -p REGEXP
List system information, check installed packages, or query package database.

At least one command option or a PROGRAM is required, as shown above.

  PROGRAM              list library (DLL) dependencies of PROGRAM
  -c, --check-setup    show installed version of PACKAGE and verify integrity
                       (or for all installed packages if none specified)
  -d, --dump-only      just list packages, do not verify (with -c)
  -s, --sysinfo        produce diagnostic system information (implies -c -d)
  -r, --registry       also scan registry for Cygwin settings (with -s)
  -k, --keycheck       perform a keyboard check session (must be run from a
                       plain console only, not from a pty/rxvt/xterm)
  -f, --find-package   find the package to which FILE belongs
  -l, --list-package   list contents of PACKAGE (or all packages if none given)
  -p, --package-query  search for REGEXP in the entire cygwin.com package
                       repository (requires internet connectivity)
  -v, --verbose        produce more verbose output
  -h, --help           annotate output with explanatory comments when given
                       with another command, otherwise print this help
  -V, --version        print the version of cygcheck and exit

Note: -c, -f, and -l only report on packages that are currently installed. To
  search all official Cygwin packages use -p instead.  The -p REGEXP matches
  package names, descriptions, and names of files/paths within all packages.
```

The **cygcheck** program is a diagnostic utility for dealing with Cygwin programs. If you are familiar with **dpkg** or **rpm**, **cygcheck** is similar in many ways. (The major difference is that **setup.exe** handles installing and uninstalling packages; see Section 2.1 for more information.)

The `-c` option checks the version and status of installed Cygwin packages. If you specify one or more package names, **cygcheck** will limit its output to those packages, or with no arguments it lists all packages. A package will be marked `Incomplete` if files originally installed are no longer present. The best thing to do in that situation is reinstall the package with **setup.exe**. To see which files are missing, use the `-v` option. If you do not need to know the status of each package and want **cygcheck** to run faster, add the `-d` option and **cygcheck** will only output the name and version for each package.

If you list one or more programs on the command line, **cygcheck** will diagnose the runtime environment of that program or programs, providing the names of DLL files on which the program depends. If you specify the `-s` option, **cygcheck** will give general system information. If you list one or more programs on the command line and specify `-s`, **cygcheck** will report on both.

The `-f` option helps you to track down which package a file came from, and `-l` lists all files in a package. For example, to find out about `/usr/bin/less` and its package:

Example 3-3. Example cygcheck usage

```
$ cygcheck -f /usr/bin/less
less-381-1

$ cygcheck -l less
/usr/bin/less.exe
/usr/bin/lessecho.exe
/usr/bin/lesskey.exe
/usr/man/man1/less.1
/usr/man/man1/lesskey.1
```

The `-h` option prints additional helpful messages in the report, at the beginning of each section. It also adds table column headings. While this is useful information, it also adds some to the size of the report, so if you want a compact report or if you know what everything is already, just leave this out.

The `-v` option causes the output to be more verbose. What this means is that additional information will be reported which is usually not interesting, such as the internal version numbers of DLLs, additional information about recursive DLL usage, and if a file in one directory in the PATH also occurs in other directories on the PATH.

The `-r` option causes **cygcheck** to search your registry for information that is relevent to Cygwin programs. These registry entries are the ones that have "Cygwin" in the name. If you are paranoid about privacy, you may remove information from this report, but please keep in mind that doing so makes it harder to diagnose your problems.

In contrast to the other options that search the packages that are installed on your local system, the `-p` option can be used to search the entire official Cygwin package repository. It takes as argument a Perl-compatible regular expression which is used to match package names, package descriptions, and path/filenames of the contents of packages. This feature requires an active internet connection, since it must query the `cygwin.com` web site. In fact, it is equalivant to the search that is available on the Cygwin package listing (http://cygwin.com/packages/) page.

For example, perhaps you are getting an error because you are missing a certain DLL and you want to know which package includes that file:

Example 3-4. Searching all packages for a file

```
$ cygcheck -p 'cygintl-2\.dll'
Found 1 matches for 'cygintl-2\.dll'.

libintl2-0.12.1-3         GNU Internationalization runtime library

$ cygcheck -p 'libexpat.*\.a'
Found 2 matches for 'libexpat.*\.a'.

expat-1.95.7-1            XML parser library written in C
expat-1.95.8-1            XML parser library written in C

$ cygcheck -p '/ls\.exe'
Found 2 matches for '/ls\.exe'.

coreutils-5.2.1-5         GNU core utilities (includes fileutils, sh-utils and textutils)
coreutils-5.3.0-6         GNU core utilities (includes fileutils, sh-utils and textutils)
```

Note that this option takes a regular expression, not a glob or wildcard. This means that you need to use `.*` if you want something similar to the wildcard `*` commonly used in filename globbing. Similarly, to match the period character you should use `\.` since the `.` character in a regexp is a metacharacter that will match any character. Also be aware that the characters such as `\` and `*` are shell metacharacters, so they must be either escaped or quoted, as in the example above.

The third example above illustrates that if you want to match a whole filename, you should include the `/` path seperator. In the given example this ensures that filenames that happen to end in `ls.exe` such as `ncftpls.exe` are not shown. Note that this use does not mean "look for packages with `ls` in the root directory," since the `/` can match anywhere in the path. It's just there to anchor the match so that it matches a full filename.

By default the matching is case-sensitive. To get a case insensitive match, begin your regexp with `(?i)` which is a PCRE-specific feature. For complete documentation on Perl-compatible regular expression syntax and options, read the **perlre** manpage, or one of many websites such as `perldoc.com` that document the Perl language.

The **cygcheck** program should be used to send information about your system for troubleshooting when requested. When asked to run this command save the output so that you can email it, for example:

```
$ cygcheck -s -v -r -h > cygcheck_output.txt
```

3.7.2. cygpath

```
Usage: cygpath (-d|-m|-u|-w|-t TYPE) [-f FILE] [OPTION]... NAME...
       cygpath [-c HANDLE]
       cygpath [-ADHOPSW]
       cygpath [-F ID]
Convert Unix and Windows format paths, or output system path information

Output type options:
  -d, --dos             print DOS (short) form of NAMEs (C:\PROGRA~1\)
  -m, --mixed           like --windows, but with regular slashes (C:/WINNT)
  -M, --mode            report on mode of file (currently binmode or textmode)
  -u, --unix            (default) print Unix form of NAMEs (/cygdrive/c/winnt)
  -w, --windows         print Windows form of NAMEs (C:\WINNT)
  -t, --type TYPE       print TYPE form: 'dos', 'mixed', 'unix', or 'windows'
Path conversion options:
  -a, --absolute        output absolute path
  -l, --long-name       print Windows long form of NAMEs (with -w, -m only)
  -p, --path            NAME is a PATH list (i.e., '/bin:/usr/bin')
  -s, --short-name      print DOS (short) form of NAMEs (with -w, -m only)
  -C, --codepage CP     print DOS, Windows, or mixed pathname in Windows
                        codepage CP.  CP can be a numeric codepage identifier,
                        or one of the reserved words ANSI, OEM, or UTF8.
                        If this option is missing, cygpath defaults to the
                        character set defined by the current locale.
System information:
  -A, --allusers        use `All Users' instead of current user for -D, -P
  -D, --desktop         output `Desktop' directory and exit
  -H, --homeroot        output `Profiles' directory (home root) and exit
  -O, --mydocs          output `My Documents' directory and exit
  -P, --smprograms      output Start Menu `Programs' directory and exit
  -S, --sysdir          output system directory and exit
  -W, --windir          output `Windows' directory and exit
  -F, --folder ID       output special folder with numeric ID and exit
Other options:
  -f, --file FILE       read FILE for input; use - to read from STDIN
  -o, --option          read options from FILE as well (for use with --file)
  -c, --close HANDLE    close HANDLE (for use in captured process)
  -i, --ignore          ignore missing argument
  -h, --help            output usage information and exit
  -v, --version         output version information and exit
```

The **cygpath** program is a utility that converts Windows native filenames to Cygwin POSIX-style pathnames and vice versa. It can be used when a Cygwin program needs to pass a file name to a native

Windows program, or expects to get a file name from a native Windows program. Alternatively, **cygpath** can output information about the location of important system directories in either format.

The `-u` and `-w` options indicate whether you want a conversion to UNIX (POSIX) format (`-u`) or to Windows format (`-w`). Use the `-d` to get DOS-style (8.3) file and path names. The `-m` option will output Windows-style format but with forward slashes instead of backslashes. This option is especially useful in shell scripts, which use backslashes as an escape character.

In combination with the `-w` option, you can use the `-l` and `-s` options to use normal (long) or DOS-style (short) form. The `-d` option is identical to `-w` and `-s` together.

The `-C` option allows to specify a Windows codepage to print DOS and Windows paths created with one of the `-d`, `-m`, or `-w` options. The default is to use the character set of the current locale defined by one of the internationalization environment variables LC_ALL, LC_CTYPE, or LANG, see Section 2.4. This is sometimes not sufficent for interaction with native Windows tools, which might expect native, non-ASCII characters in a specific Windows codepage. Console tools, for instance, might expect pathnames in the current OEM codepage, while graphical tools like Windows Explorer might expect pathnames in the current ANSI codepage.

The `-C` option takes a single parameter:

- `ANSI`, to specify the current ANSI codepage
- `OEM`, to specify the current OEM (console) codepage
- `UTF8`, to specify UTF-8.
- A numerical, decimal codepage number, for instance 936 for GBK, 28593 for ISO-8859-3, etc. A full list of supported codepages is listed on the Microsoft MSDN page Code Page Identifiers (http://msdn.microsoft.com/en-us/library/dd317756(VS.85).aspx). A codepage of 0 is the same as if the `-C` hasn't been specified at all.

The `-p` option means that you want to convert a path-style string rather than a single filename. For example, the PATH environment variable is semicolon-delimited in Windows, but colon-delimited in UNIX. By giving `-p` you are instructing **cygpath** to convert between these formats.

The `-i` option supresses the print out of the usage message if no filename argument was given. It can be used in make file rules converting variables that may be omitted to a proper format. Note that **cygpath** output may contain spaces (C:\Program Files) so should be enclosed in quotes.

Example 3-5. Example cygpath usage

```
#!/bin/sh
if [ "${1}" = "" ];
 then
  XPATH=".";
 else
  XPATH="$(cygpath -C ANSI -w "${1}")";
```

```
fi
explorer $XPATH &
```

The capital options `-D`, `-H`, `-P`, `-S`, and `-W` output directories used by Windows that are not the same on all systems, for example `-S` might output C:\WINNT\system32 or C:\Windows\System32. The `-H` shows the Windows profiles directory that can be used as root of home. The `-A` option forces use of the "All Users" directories instead of the current user for the `-D`, `-O` and `-P` options. The `-F` outputs other special folders specified by their internal numeric code (decimal or 0xhex). For valid codes and symbolic names, see the CSIDL_* definitions in the include file /usr/include/w32api/shlobj.h from package w32api. The current valid range of codes for folders is 0 (Desktop) to 59 (CDBurn area). By default the output is in UNIX (POSIX) format; use the `-w` or `-d` options to get other formats.

3.7.3. dumper

```
Usage: dumper [OPTION] FILENAME WIN32PID
Dump core from WIN32PID to FILENAME.core

-d, --verbose  be verbose while dumping
-h, --help     output help information and exit
-q, --quiet    be quiet while dumping (default)
-v, --version  output version information and exit
```

The **dumper** utility can be used to create a core dump of running Windows process. This core dump can be later loaded to **gdb** and analyzed. One common way to use **dumper** is to plug it into cygwin's Just-In-Time debugging facility by adding

```
error_start=x:\path\to\dumper.exe
```

to the *CYGWIN* environment variable. Please note that `x:\path\to\dumper.exe` is Windows-style and not cygwin path. If `error_start` is set this way, then dumper will be started whenever some program encounters a fatal error.

dumper can be also be started from the command line to create a core dump of any running process. Unfortunately, because of a Windows API limitation, when a core dump is created and **dumper** exits, the target process is terminated too.

To save space in the core dump, **dumper** doesn't write those portions of target process' memory space that are loaded from executable and dll files and are unchangeable, such as program code and debug info. Instead, **dumper** saves paths to files which contain that data. When a core dump is loaded into gdb, it uses these paths to load appropriate files. That means that if you create a core dump on one machine and try to debug it on another, you'll need to place identical copies of the executable and dlls in the same directories as on the machine where the core dump was created.

3.7.4. getfacl

```
Usage: getfacl [-adn] FILE [FILE2...]
Display file and directory access control lists (ACLs).

  -a, --all      display the filename, the owner, the group, and
                 the ACL of the file
  -d, --dir      display the filename, the owner, the group, and
                 the default ACL of the directory, if it exists
  -h, --help     output usage information and exit
  -n, --noname   display user and group IDs instead of names
  -v, --version  output version information and exit

When multiple files are specified on the command line, a blank
line separates the ACLs for each file.
```

For each argument that is a regular file, special file or directory, **getfacl** displays the owner, the group, and the ACL. For directories **getfacl** displays additionally the default ACL. With no options specified, **getfacl** displays the filename, the owner, the group, and both the ACL and the default ACL, if it exists. For more information on Cygwin and Windows ACLs, see Section 2.5 in the Cygwin User's Guide. The format for ACL output is as follows:

```
# file: filename
# owner: name or uid
# group: name or uid
user::perm
user:name or uid:perm
group::perm
group:name or gid:perm
mask:perm
other:perm
default:user::perm
default:user:name or uid:perm
default:group::perm
default:group:name or gid:perm
default:mask:perm
default:other:perm
```

3.7.5. kill

```
Usage: kill [-f] [-signal] [-s signal] pid1 [pid2 ...]
       kill -l [signal]
Send signals to processes

 -f, --force     force, using win32 interface if necessary
 -l, --list      print a list of signal names
 -s, --signal    send signal (use kill --list for a list)
 -h, --help      output usage information and exit
 -v, --version   output version information and exit
```

The **kill** program allows you to send arbitrary signals to other Cygwin programs. The usual purpose is to end a running program from some other window when ^C won't work, but you can also send program-specified signals such as SIGUSR1 to trigger actions within the program, like enabling debugging or re-opening log files. Each program defines the signals they understand.

You may need to specify the full path to use **kill** from within some shells, including **bash**, the default Cygwin shell. This is because **bash** defines a **kill** builtin function; see the **bash** man page under *BUILTIN COMMANDS* for more information. To make sure you are using the Cygwin version, try

```
$ /bin/kill --version
```

which should give the Cygwin **kill** version number and copyright information.

Unless you specific the `-f` option, the "pid" values used by **kill** are the Cygwin pids, not the Windows pids. To get a list of running programs and their Cygwin pids, use the Cygwin **ps** program. **ps -W** will display *all* windows pids.

The **kill -l** option prints the name of the given signal, or a list of all signal names if no signal is given.

To send a specific signal, use the `-signN` option, either with a signal number or a signal name (minus the "SIG" part), as shown in these examples:

Example 3-6. Using the kill command

```
$ kill 123
$ kill -1 123
$ kill -HUP 123
$ kill -f 123
```

Here is a list of available signals, their numbers, and some commentary on them, from the file `<sys/signal.h>`, which should be considered the official source of this information.

```
SIGHUP       1    hangup
SIGINT       2    interrupt
SIGQUIT      3    quit
SIGILL       4    illegal instruction (not reset when caught)
SIGTRAP      5    trace trap (not reset when caught)
SIGABRT      6    used by abort
SIGEMT       7    EMT instruction
SIGFPE       8    floating point exception
SIGKILL      9    kill (cannot be caught or ignored)
SIGBUS      10    bus error
SIGSEGV     11    segmentation violation
SIGSYS      12    bad argument to system call
SIGPIPE     13    write on a pipe with no one to read it
SIGALRM     14    alarm clock
SIGTERM     15    software termination signal from kill
SIGURG      16    urgent condition on IO channel
```

```
SIGSTOP     17    sendable stop signal not from tty
SIGTSTP     18    stop signal from tty
SIGCONT     19    continue a stopped process
SIGCHLD     20    to parent on child stop or exit
SIGTTIN     21    to readers pgrp upon background tty read
SIGTTOU     22    like TTIN for output if (tp->t_local&LTOSTOP)
SIGPOLL     23    System V name for SIGIO
SIGXCPU     24    exceeded CPU time limit
SIGXFSZ     25    exceeded file size limit
SIGVTALRM   26    virtual time alarm
SIGPROF     27    profiling time alarm
SIGWINCH    28    window changed
SIGLOST     29    resource lost (eg, record-lock lost)
SIGUSR1     30    user defined signal 1
SIGUSR2     31    user defined signal 2
```

3.7.6. mkgroup

```
Usage: mkgroup [OPTION]...
Print /etc/group file to stdout

Options:
   -l,--local [machine[,offset]]
                           print local groups with gid offset offset
                           (from local machine if no machine specified)
   -L,--Local [machine[,offset]]
                           ditto, but generate groupname with machine prefix
   -d,--domain [domain[,offset]]
                           print domain groups with gid offset offset
                           (from current domain if no domain specified)
   -D,--Domain [domain[,offset]]
                           ditto, but generate groupname with machine prefix
   -c,--current            print current group
   -C,--Current            ditto, but generate groupname with machine or
                           domain prefix
   -S,--separator char     for -L, -D, -C use character char as domain\group
                           separator in groupname instead of the default '\'
   -o,--id-offset offset   change the default offset (10000) added to gids
                           in domain or foreign server accounts.
   -g,--group groupname    only return information for the specified group
                           one of -l, -L, -d, -D must be specified, too
   -b,--no-builtin         don't print BUILTIN groups
   -U,--unix grouplist     additionally print UNIX groups when using -l or -L
                           on a UNIX Samba server
                           grouplist is a comma-separated list of groupnames
                           or gid ranges (root,-25,50-100).
      (enumerating large ranges can take a long time!)
   -s,--no-sids            (ignored)
   -u,--users              (ignored)
   -h,--help               print this message
   -v,--version            print version information and exit
```

```
Default is to print local groups on stand-alone machines, plus domain
groups on domain controllers and domain member machines.
```

The **mkgroup** program can be used to help configure Cygwin by creating a `/etc/group` file. Its use is essential to include Windows security information.

The command is initially called by **setup.exe** to create a default `/etc/group`. This should be sufficient in most circumstances. However, especially when working in a multi-domain environment, you can use **mkgroup** manually to create a more complete `/etc/group` file for all domains. Especially when you have the same group name used on multiple machines or in multiple domains, you can use the `-D`, `-L` and `-C` options to create unique domain\group style groupnames.

Note that this information is static. If you change the group information in your system, you'll need to regenerate the group file for it to have the new information.

The `-d/-D` and `-l/-L` options allow you to specify where the information comes from, the local SAM of a machine or from the domain, or both. With the `-d/-D` options the program contacts a Domain Controller, which my be unreachable or have restricted access. Comma-separated from the machine or domain, you can specify an offset which is used as base added to the group's RID to compute the gid (offset + RID = gid). This allows you to create the same gids every time you re-run **mkgroup**. For very simple needs, an entry for the current user's group can be created by using the option `-c` or `-C`. If you want to use one of the `-D`, `-L` or `-C` options, but you don't like the backslash as domain/group separator, you can specify another separator using the `-S` option, for instance:

Example 3-7. Setting up group entry for current user with different domain/group separator

```
$ mkgroup -C -S+ > /etc/group
$ cat /etc/group
DOMAIN+my_group:S-1-5-21-2913048732-1697188782-3448811101-1144:11144:
```

The `-o` option allows for special cases (such as multiple domains) where the GIDs might match otherwise. The `-g` option only prints the information for one group. The `-U` option allows you to enumerate the standard UNIX groups on a Samba machine. It's used together with `-l samba-server` or `-L samba-server`. The normal UNIX groups are usually not enumerated, but they can show up as a group in **ls -l** output.

3.7.7. mkpasswd

```
Usage: mkpasswd [OPTIONS]...
Print /etc/passwd file to stdout

Options:
   -l,--local [machine[,offset]]
                           print local user accounts with uid offset offset
                           (from local machine if no machine specified)
```

```
   -L,--Local [machine[,offset]]
                            ditto, but generate username with machine prefix
   -d,--domain [domain[,offset]]
                            print domain accounts with uid offset offset
                            (from current domain if no domain specified)
   -D,--Domain [domain[,offset]]
                            ditto, but generate username with domain prefix
   -c,--current             print current user
   -C,--Current             ditto, but generate username with machine or
                            domain prefix
   -S,--separator char      for -L, -D, -C use character char as domain\user
                            separator in username instead of the default '\'
   -o,--id-offset offset    change the default offset (10000) added to uids
                            in domain or foreign server accounts.
   -u,--username username   only return information for the specified user
                            one of -l, -L, -d, -D must be specified, too
   -p,--path-to-home path   use specified path instead of user account home dir
                            or /home prefix
   -m,--no-mount            don't use mount points for home dir
   -U,--unix userlist       additionally print UNIX users when using -l or -L\
                            on a UNIX Samba server
                            userlist is a comma-separated list of usernames
                            or uid ranges (root,-25,50-100).
                            (enumerating large ranges can take a long time!)
   -s,--no-sids             (ignored)
   -g,--local-groups        (ignored)
   -h,--help                displays this message
   -v,--version             version information and exit

Default is to print local accounts on stand-alone machines, domain accounts
on domain controllers and domain member machines.
```

The **mkpasswd** program can be used to help configure Cygwin by creating a `/etc/passwd` from your system information. Its use is essential to include Windows security information. However, the actual passwords are determined by Windows, not by the content of `/etc/passwd`.

The command is initially called by **setup.exe** to create a default `/etc/passwd`. This should be sufficient in most circumstances. However, especially when working in a multi-domain environment, you can use **mkpasswd** manually to create a more complete `/etc/passwd` file for all domains. Especially when you have the same user name used on multiple machines or in multiple domains, you can use the `-D`, `-L` and `-C` options to create unique domain\user style usernames.

Note that this information is static. If you change the user information in your system, you'll need to regenerate the passwd file for it to have the new information.

The `-d/-D` and `-l/-L` options allow you to specify where the information comes from, the local machine or the domain (default or given), or both. With the `-d/-D` options the program contacts the Domain Controller, which may be unreachable or have restricted access. Comma-separated from the machine or domain, you can specify an offset which is used as base added to the user's RID to compute the uid (offset + RID = uid). This allows to create the same uids every time you re-run **mkpasswd**. An entry for the current user can be created by using the option `-c` or `-C`. If you want to use one of the `-D`, `-L` or `-C` options, but you don't like the backslash as domain/group separator, you can specify another separator using the `-S` option, similar to the **mkgroup**. The `-o` option allows for special cases (such as multiple domains) where the UIDs might match otherwise. The `-m` option bypasses the current mount table so that, for example, two users who have a Windows home directory of H: could mount them differently. For more information on SIDs, see Section 2.5 in the Cygwin User's Guide. The `-p` option causes **mkpasswd** to use the specified prefix instead of the account home dir or `/home/` . For example, this command:

Example 3-8. Using an alternate home root

```
$ mkpasswd -l -p "$(cygpath -H)" > /etc/passwd
```

would put local users' home directories in the Windows 'Profiles' directory. The `-u` option creates just an entry for the specified user. The `-U` option allows you to enumerate the standard UNIX users on a Samba machine. It's used together with `-l samba-server` or `-L samba-server`. The normal UNIX users are usually not enumerated, but they can show up as file owners in **ls -l** output.

3.7.8. mount

```
Usage: mount [OPTION] [<win32path> <posixpath>]
       mount -a
       mount <posixpath>
Display information about mounted filesystems, or mount a filesystem

  -a, --all                     mount all filesystems mentioned in fstab
  -c, --change-cygdrive-prefix  change the cygdrive path prefix to <posixpath>
  -f, --force                   force mount, don't warn about missing mount
                                point directories
  -h, --help                    output usage information and exit
  -m, --mount-entries           write fstab entries to replicate mount points
                                and cygdrive prefixes
  -o, --options X[,X...]        specify mount options
  -p, --show-cygdrive-prefix    show user and/or system cygdrive path prefix
  -v, --version                 output version information and exit
```

The **mount** program is used to map your drives and shares onto Cygwin's simulated POSIX directory tree, much like as is done by mount commands on typical UNIX systems. However, in contrast to mount points given in `/etc/fstab`, mount points created or changed with **mount** are not persistent. They disappear immediately after the last process of the current user exited. Please see Section 3.1.2 for more information on the concepts behind the Cygwin POSIX file system and strategies for using mounts. To remove mounts temporarily, use **umount**

3.7.8.1. Using mount

If you just type **mount** with no parameters, it will display the current mount table for you.

Example 3-9. Displaying the current set of mount points

```
$ mount
c:\cygwin\bin on /usr/bin type ntfs (binary)
c:\cygwin\lib on /usr/lib type ntfs (binary)
c:\cygwin on / type ntfs (binary)
c: on /c type ntfs (binary,user,noumount)
d: on /d type fat (binary,user,noumount)
```

In this example, c:\cygwin is the POSIX root and D drive is mapped to `/d`. Note that in this case, the root mount is a system-wide mount point that is visible to all users running Cygwin programs, whereas the `/d` mount is only visible to the current user.

The **mount** utility is also the mechanism for adding new mounts to the mount table. The following example demonstrates how to mount the directory `//pollux/home/joe/data` to `/data` for the duration of the current session.

Example 3-10. Adding mount points

```
$ ls /data
ls: /data: No such file or directory
$ mount //pollux/home/joe/data /data
mount: warning - /data does not exist!
$ mount
//pollux/home/joe/data on /data type smbfs (binary)
c:/cygwin/bin on /usr/bin type ntfs (binary)
c:/cygwin/lib on /usr/lib type ntfs (binary)
c:/cygwin on / type ntfs (binary)
c: on /c type ntfs (binary,user,noumount)
d: on /d type fat (binary,user,noumount)
```

A given POSIX path may only exist once in the mount table. Attempts to replace the mount will fail with a busy error. The `-f` (force) option causes the old mount to be silently replaced with the new one, provided the old mount point was a user mount point. It's not valid to replace system-wide mount points. Additionally, the `-f` option will silence warnings about the non-existence of directories at the Win32 path location.

The `-o` option is the method via which various options about the mount point may be recorded. The following options are available (note that most of the options are duplicates of other mount flags):

```
  acl        - Use the filesystem's access control lists (ACLs) to
               implement real POSIX permissions (default).
  binary     - Files default to binary mode (default).
  cygexec    - Treat all files below mount point as cygwin executables.
  exec       - Treat all files below mount point as executable.
  noacl      - Ignore ACLs and fake POSIX permissions.
  nosuid     - No suid files are allowed (currently unimplemented)
  notexec    - Treat all files below mount point as not executable.
  override   - Override immutable mount points.
```

```
posix=0    - Switch off case sensitivity for paths under this mount point.
posix=1    - Switch on case sensitivity for paths under this mount point
              (default).
text       - Files default to CRLF text mode line endings.
```

For a more complete description of the mount options and the `/etc/fstab` file, see Section 3.1.2.

Note that all mount points added with **mount** are user mount points. System mount points can only be specified in the `/etc/fstab` file.

If you added mount points to `/etc/fstab` or your `/etc/fstab.d/<username>` file, you can add these mount points to your current user session using the `-a/--all` option, or by specifing the posix path alone on the command line. As an example, consider you added a mount point with the POSIX path `/my/mount`. You can add this mount point with either one of the following two commands to your current user session.

```
$ mount /my/mount
$ mount -a
```

The first command just adds the `/my/mount` mount point to your current session, the **mount -a** adds all new mount points to your user session.

If you change a mount point to point to another native path, or if you changed the flags of a mount point, you have to **umount** the mount point first, before you can add it again. Please note that all such added mount points are added as user mount points, and that the rule that system mount points can't be removed or replaced in a running session still applies.

The `-m` option causes the **mount** utility to output the current mount table in a series of fstab entries. You can save this output as a backup when experimenting with the mount table. Copy the output to `/etc/fstab` to restore the old state. It also makes moving your settings to a different machine much easier.

3.7.8.2. Cygdrive mount points

Whenever Cygwin cannot use any of the existing mounts to convert from a particular Win32 path to a POSIX one, Cygwin will, instead, convert to a POSIX path using a default mount point: `/cygdrive`. For example, if Cygwin accesses `z:\foo` and the z drive is not currently in the mount table, then `z:\` will be accessible as `/cygdrive/z`. The **mount** utility can be used to change this default automount prefix through the use of the "--change-cygdrive-prefix" option. In the following example, we will set the automount prefix to `/mnt`:

Example 3-11. Changing the default prefix

```
$ mount --change-cygdrive-prefix /mnt
```

Note that the cygdrive prefix can be set both per-user and system-wide, and that as with all mounts, a user-specific mount takes precedence over the system-wide setting. The **mount** utility creates system-wide mounts by default if you do not specify a type. You can always see the user and system cygdrive prefixes with the `-p` option. Using the `--options` flag with `--change-cygdrive-prefix` makes all new automounted filesystems default to this set of options. For instance (using the short form of the command line flags)

Example 3-12. Changing the default prefix with specific mount options

```
$ mount -c /mnt -o binary,noacl
```

3.7.8.3. Limitations

Limitations: there is a hard-coded limit of 30 mount points. Also, although you can mount to pathnames that do not start with "/", there is no way to make use of such mount points.

Normally the POSIX mount point in Cygwin is an existing empty directory, as in standard UNIX. If this is the case, or if there is a place-holder for the mount point (such as a file, a symbolic link pointing anywhere, or a non-empty directory), you will get the expected behavior. Files present in a mount point directory before the mount become invisible to Cygwin programs.

It is sometimes desirable to mount to a non-existent directory, for example to avoid cluttering the root directory with names such as `a`, `b`, `c` pointing to disks. Although **mount** will give you a warning, most everything will work properly when you refer to the mount point explicitly. Some strange effects can occur however. For example if your current working directory is `/dir`, say, and `/dir/mtpt` is a mount point, then `mtpt` will not show up in an **ls** or **echo *** command and **find .** will not find `mtpt`.

3.7.9. passwd

```
Usage: passwd [OPTION] [USER]
Change USER's password or password attributes.

User operations:
  -l, --lock               lock USER's account.
  -u, --unlock             unlock USER's account.
  -c, --cannot-change      USER can't change password.
  -C, --can-change         USER can change password.
  -e, --never-expires      USER's password never expires.
  -E, --expires            USER's password expires according to system's
                           password aging rule.
  -p, --pwd-not-required   no password required for USER.
  -P, --pwd-required       password is required for USER.
  -R, --reg-store-pwd      enter password to store it in the registry for
                           later usage by services to be able to switch
                           to this user context with network credentials.
```

```
System operations:
  -i, --inactive NUM       set NUM of days before inactive accounts are disabled
                           (inactive accounts are those with expired passwords).
  -n, --minage DAYS        set system minimum password age to DAYS days.
  -x, --maxage DAYS        set system maximum password age to DAYS days.
  -L, --length LEN         set system minimum password length to LEN.

Other options:
  -d, --logonserver SERVER connect to SERVER (e.g. domain controller).
                           Default server is the local system, unless
                           changing the current user, in which case the
                           default is the content of $LOGONSERVER.
  -S, --status             display password status for USER (locked, expired,
                           etc.) plus global system password settings.
  -h, --help               output usage information and exit.
  -v, --version            output version information and exit.

If no option is given, change USER's password.  If no user name is given,
operate on current user.  System operations must not be mixed with user
operations.  Don't specify a USER when triggering a system operation.

Don't specify a user or any other option together with the -R option.
Non-Admin users can only store their password if cygserver is running.
Note that storing even obfuscated passwords in the registry is not overly
secure.  Use this feature only if the machine is adequately locked down.
Don't use this feature if you don't need network access within a remote
session.  You can delete your stored password by using 'passwd -R' and
specifying an empty password.
```

passwd changes passwords for user accounts. A normal user may only change the password for their own account, but administrators may change passwords on any account. **passwd** also changes account information, such as password expiry dates and intervals.

For password changes, the user is first prompted for their old password, if one is present. This password is then encrypted and compared against the stored password. The user has only one chance to enter the correct password. The administrators are permitted to bypass this step so that forgotten passwords may be changed.

The user is then prompted for a replacement password. **passwd** will prompt twice for this replacement and compare the second entry against the first. Both entries are required to match in order for the password to be changed.

After the password has been entered, password aging information is checked to see if the user is permitted to change their password at this time. If not, **passwd** refuses to change the password and exits.

To get current password status information, use the `-S` option. Administrators can use **passwd** to perform several account maintenance functions (users may perform some of these functions on their own accounts). Accounts may be locked with the `-l` flag and unlocked with the `-u` flag. Similarly, `-c` disables a user's ability to change passwords, and `-C` allows a user to change passwords. For password expiry, the `-e` option disables expiration, while the `-E` option causes the password to expire according to the system's normal aging rules. Use `-p` to disable the password requirement for a user, or `-P` to require a password.

Administrators can also use **passwd** to change system-wide password expiry and length requirements with the `-i`, `-n`, `-x`, and `-L` options. The `-i` option is used to disable an account after the password has been expired for a number of days. After a user account has had an expired password for *NUM* days, the user may no longer sign on to the account. The `-n` option is used to set the minimum number of days before a password may be changed. The user will not be permitted to change the password until *MINDAYS* days have elapsed. The `-x` option is used to set the maximum number of days a password remains valid. After *MAXDAYS* days, the password is required to be changed. Allowed values for the above options are 0 to 999. The `-L` option sets the minimum length of allowed passwords for users who don't belong to the administrators group to *LEN* characters. Allowed values for the minimum password length are 0 to 14. In any of the above cases, a value of 0 means 'no restrictions'.

All operations affecting the current user are by default run against the logon server of the current user (taken from the environment variable LOGONSERVER. When password or account information of other users should be changed, the default server is the local system. To change a user account on a remote machine, use the `-d` option to specify the machine to run the command against. Note that the current user must be a valid member of the administrators group on the remote machine to perform such actions.

Users can use the **passwd -R** to enter a password which then gets stored in a special area of the registry on the local system, which is also used by Windows to store passwords of accounts running Windows services. When a privileged Cygwin application calls the **set{e}uid(user_id)** system call, Cygwin checks if a password for that user has been stored in this registry area. If so, it uses this password to switch to this user account using that password. This allows you to logon through, for instance, **ssh** with public key authentication and get a full qualified user token with all credentials for network access. However, the method has some drawbacks security-wise. This is explained in more detail in Section 2.5.

Please note that storing passwords in that registry area is a privileged operation which only administrative accounts are allowed to do. If normal, non-admin users should be allowed to enter their passwords using **passwd -R**, it's required to run **cygserver** as a service under the LocalSystem account before running **passwd -R**. This only affects storing passwords. Using passwords in privileged processes does not require **cygserver** to run.

Limitations: Users may not be able to change their password on some systems.

3.7.10. ps

```
Usage: ps [-aefls] [-u UID]
Report process status

 -a, --all       show processes of all users
```

```
 -e, --everyone  show processes of all users
 -f, --full      show process uids, ppids
 -h, --help      output usage information and exit
 -l, --long      show process uids, ppids, pgids, winpids
 -p, --process   show information for specified PID
 -s, --summary   show process summary
 -u, --user      list processes owned by UID
 -v, --version   output version information and exit
 -W, --windows   show windows as well as cygwin processes
With no options, ps outputs the long format by default
```

The **ps** program gives the status of all the Cygwin processes running on the system (ps = "process status"). Due to the limitations of simulating a POSIX environment under Windows, there is little information to give.

The PID column is the process ID you need to give to the **kill** command. The PPID is the parent process ID, and PGID is the process group ID. The WINPID column is the process ID displayed by NT's Task Manager program. The TTY column gives which pseudo-terminal a process is running on, or a `'?'` for services. The UID column shows which user owns each process. STIME is the time the process was started, and COMMAND gives the name of the program running. Listings may also have a status flag in column zero; `S` means stopped or suspended (in other words, in the background), `I` means waiting for input or interactive (foreground), and `O` means waiting to output.

By default, **ps** will only show processes owned by the current user. With either the `-a` or `-e` option, all user's processes (and system processes) are listed. There are historical UNIX reasons for the synonomous options, which are functionally identical. The `-f` option outputs a "full" listing with usernames for UIDs. The `-l` option is the default display mode, showing a "long" listing with all the above columns. The other display option is `-s`, which outputs a shorter listing of just PID, TTY, STIME, and COMMAND. The `-u` option allows you to show only processes owned by a specific user. The `-p` option allows you to show information for only the process with the specified PID. The `-W` option causes **ps** show non-Cygwin Windows processes as well as Cygwin processes. The WINPID is also the PID, and they can be killed with the Cygwin **kill** command's `-f` option.

3.7.11. regtool

```
Usage: regtool [OPTION] (add|check|get|list|remove|unset|load|unload|save) KEY
View or edit the Win32 registry

Actions:
 add KEY\SUBKEY             add new SUBKEY
 check KEY                  exit 0 if KEY exists, 1 if not
 get KEY\VALUE              prints VALUE to stdout
 list KEY                   list SUBKEYs and VALUEs
 remove KEY                 remove KEY
 set KEY\VALUE [data ...]   set VALUE
 unset KEY\VALUE            removes VALUE from KEY
 load KEY\SUBKEY PATH       load hive from PATH into new SUBKEY
```

```
 unload KEY\SUBKEY          unload hive and remove SUBKEY
 save KEY\SUBKEY PATH       save SUBKEY into new hive PATH

Options for 'list' Action:
 -k, --keys           print only KEYs
 -l, --list           print only VALUEs
 -p, --postfix        like ls -p, appends '\' postfix to KEY names

Options for 'get' Action:
 -b, --binary         print REG_BINARY data as hex bytes

Options for 'set' Action:
 -b, --binary         set type to REG_BINARY (hex args or '-')
 -e, --expand-string  set type to REG_EXPAND_SZ
 -i, --integer        set type to REG_DWORD
 -m, --multi-string   set type to REG_MULTI_SZ
 -s, --string         set type to REG_SZ

Options for 'set' and 'unset' Actions:
 -K<c>, --key-separator[=]<c>  set key separator to <c> instead of '\'

Other Options:
 -h, --help     output usage information and exit
 -q, --quiet    no error output, just nonzero return if KEY/VALUE missing
 -v, --verbose  verbose output, including VALUE contents when applicable
 -w, --wow64    access 64 bit registry view (ignored on 32 bit Windows)
 -W, --wow32    access 32 bit registry view (ignored on 32 bit Windows)
 -V, --version  output version information and exit

KEY is in the format [host]\prefix\KEY\KEY\VALUE, where host is optional
remote host in either \\hostname or hostname: format and prefix is any of:
  root     HKCR  HKEY_CLASSES_ROOT (local only)
  config   HKCC  HKEY_CURRENT_CONFIG (local only)
  user     HKCU  HKEY_CURRENT_USER (local only)
  machine  HKLM  HKEY_LOCAL_MACHINE
  users    HKU   HKEY_USERS

You can use forward slash ('/') as a separator instead of backslash, in
that case backslash is treated as escape character
Example: regtool.exe get '\user\software\Microsoft\Clock\iFormat'
```

The **regtool** program allows shell scripts to access and modify the Windows registry. Note that modifying the Windows registry is dangerous, and carelessness here can result in an unusable system. Be careful.

The `-v` option means "verbose". For most commands, this causes additional or lengthier messages to be printed. Conversely, the `-q` option supresses error messages, so you can use the exit status of the program to detect if a key exists or not (for example).

The `-w` option allows you to access the 64 bit view of the registry. Several subkeys exist in a 32 bit and a 64 bit version when running on Windows 64. Since Cygwin is running in 32 bit mode, it only has access to the 32 bit view of these registry keys. When using the `-w` switch, the 64 bit view is used and **regtool** can access the entire registry. This option is simply ignored when running on 32 bit Windows versions.

The `-W` option allows you to access the 32 bit view on the registry. The purpose of this option is mainly for symmetry. It permits creation of OS agnostic scripts which would also work in a hypothetical 64 bit version of Cygwin.

You must provide **regtool** with an *action* following options (if any). Currently, the action must be `add`, `set`, `check`, `get`, `list`, `remove`, `set`, or `unset`.

The `add` action adds a new key. The `check` action checks to see if a key exists (the exit code of the program is zero if it does, nonzero if it does not). The `get` action gets the value of a key, and prints it (and nothing else) to stdout. Note: if the value doesn't exist, an error message is printed and the program returns a non-zero exit code. If you give `-q`, it doesn't print the message but does return the non-zero exit code.

The `list` action lists the subkeys and values belonging to the given key. With `list`, the `-k` option instructs **regtool** to print only KEYs, and the `-l` option to print only VALUEs. The `-p` option postfixes a `'/'` to each KEY, but leave VALUEs with no postfix. The `remove` action removes a key. Note that you may need to remove everything in the key before you may remove it, but don't rely on this stopping you from accidentally removing too much.

The `set` action sets a value within a key. `-b` means it's binary data (REG_BINARY). The binary values are specified as hex bytes in the argument list. If the argument is `'-'`, binary data is read from stdin instead. `-e` means it's an expanding string (REG_EXPAND_SZ) that contains embedded environment variables. `-i` means the value is an integer (REG_DWORD). `-m` means it's a multi-string (REG_MULTI_SZ). `-s` means the value is a string (REG_SZ). If you don't specify one of these, **regtool** tries to guess the type based on the value you give. If it looks like a number, it's a DWORD. If it starts with a percent, it's an expanding string. If you give multiple values, it's a multi-string. Else, it's a regular string. The `unset` action removes a value from a key.

The `load` action adds a new subkey and loads the contents of a registry hive into it. The parent key must be HKEY_LOCAL_MACHINE or HKEY_USERS. The `unload` action unloads the file and removes the subkey.

The `save` action saves a subkey into a registry hive.

By default, the last "\" or "/" is assumed to be the separator between the key and the value. You can use the `-K` option to provide an alternate key/value separator character.

3.7.12. setfacl

```
Usage: setfacl [-r] (-f ACL_FILE | -s acl_entries) FILE...
```

```
       setfacl [-r] ([-d acl_entries] [-m acl_entries]) FILE...
Modify file and directory access control lists (ACLs)

  -d, --delete     delete one or more specified ACL entries
  -f, --file       set ACL entries for FILE to ACL entries read
                   from a ACL_FILE
  -m, --modify     modify one or more specified ACL entries
  -r, --replace    replace mask entry with maximum permissions
                   needed for the file group class
  -s, --substitute substitute specified ACL entries for the
                   ACL of FILE
  -h, --help       output usage information and exit
  -v, --version    output version information and exit

At least one of (-d, -f, -m, -s) must be specified
```

For each file given as parameter, **setfacl** will either replace its complete ACL (`-s`, `-f`), or it will add, modify, or delete ACL entries. For more information on Cygwin and Windows ACLs, see see Section 2.5 in the Cygwin User's Guide.

Acl_entries are one or more comma-separated ACL entries from the following list:

```
u[ser]::perm
u[ser]:uid:perm
g[roup]::perm
g[roup]:gid:perm
m[ask]::perm
o[ther]::perm
```

Default entries are like the above with the additional default identifier. For example:

```
d[efault]:u[ser]:uid:perm
```

perm is either a 3-char permissions string in the form "rwx" with the character `'-'` for no permission or it is the octal representation of the permissions, a value from 0 (equivalent to "---") to 7 ("rwx"). *uid* is a user name or a numerical uid. *gid* is a group name or a numerical gid.

The following options are supported:

`-d` Delete one or more specified entries from the file's ACL. The owner, group and others entries must not be deleted. Acl_entries to be deleted should be specified without permissions, as in the following list:

```
u[ser]:uid
g[roup]:gid
d[efault]:u[ser]:uid
d[efault]:g[roup]:gid
d[efault]:m[ask]:
```

```
d[efault]:o[ther]:
```

`-f` Take the Acl_entries from ACL_FILE one per line. Whitespace characters are ignored, and the character "#" may be used to start a comment. The special filename "-" indicates reading from stdin. Note that you can use this with **getfacl** and **setfacl** to copy ACLs from one file to another:

```
$ getfacl source_file | setfacl -f - target_file
```

Required entries are: one user entry for the owner of the file, one group entry for the group of the file, and one other entry.

If additional user and group entries are given: a mask entry for the file group class of the file, and no duplicate user or group entries with the same uid/gid.

If it is a directory: one default user entry for the owner of the file, one default group entry for the group of the file, one default mask entry for the file group class, and one default other entry.

`-m` Add or modify one or more specified ACL entries. Acl_entries is a comma-separated list of entries from the same list as above.

`-r` Causes the permissions specified in the mask entry to be ignored and replaced by the maximum permissions needed for the file group class.

`-s` Like `-f`, but substitute the file's ACL with Acl_entries specified in a comma-separated list on the command line.

While the `-d` and `-m` options may be used in the same command, the `-f` and `-s` options may be used only exclusively.

Directories may contain default ACL entries. Files created in a directory that contains default ACL entries will have permissions according to the combination of the current umask, the explicit permissions requested and the default ACL entries

Limitations: Under Cygwin, the default ACL entries are not taken into account currently.

3.7.13. ssp

```
Usage: ssp [options] low_pc high_pc command...
```

```
Single-step profile COMMAND

 -c, --console-trace  trace every EIP value to the console. *Lots* slower.
 -d, --disable        disable single-stepping by default; use
                      OutputDebugString ("ssp on") to enable stepping
 -e, --enable         enable single-stepping by default; use
                      OutputDebugString ("ssp off") to disable stepping
 -h, --help           output usage information and exit
 -l, --dll            enable dll profiling.  A chart of relative DLL usage
                      is produced after the run.
 -s, --sub-threads    trace sub-threads too.  Dangerous if you have
                      race conditions.
 -t, --trace-eip      trace every EIP value to a file TRACE.SSP.  This
                      gets big *fast*.
 -v, --verbose        output verbose messages about debug events.
 -V, --version        output version information and exit

Example: ssp 0x401000 0x403000 hello.exe
```

SSP - The Single Step Profiler

Original Author: DJ Delorie

The SSP is a program that uses the Win32 debug API to run a program one ASM instruction at a time. It records the location of each instruction used, how many times that instruction is used, and all function calls. The results are saved in a format that is usable by the profiling program **gprof**, although **gprof** will claim the values are seconds, they really are instruction counts. More on that later.

Because the SSP was originally designed to profile the cygwin DLL, it does not automatically select a block of code to report statistics on. You must specify the range of memory addresses to keep track of manually, but it's not hard to figure out what to specify. Use the "objdump" program to determine the bounds of the target's ".text" section. Let's say we're profiling cygwin1.dll. Make sure you've built it with debug symbols (else **gprof** won't run) and run objdump like this:

```
$ objdump -h cygwin1.dll
```

It will print a report like this:

```
cygwin1.dll:     file format pei-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0007ea00  61001000  61001000  00000400  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE, DATA
  1 .data         00008000  61080000  61080000  0007ee00  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  . . .
```

The only information we're concerned with are the VMA of the .text section and the VMA of the section after it (sections are usually contiguous; you can also add the Size to the VMA to get the end address). In this case, the VMA is 0x61001000 and the ending address is either 0x61080000 (start of .data method) or 0x0x6107fa00 (VMA+Size method).

There are two basic ways to use SSP - either profiling a whole program, or selectively profiling parts of the program.

To profile a whole program, just run **ssp** without options. By default, it will step the whole program. Here's a simple example, using the numbers above:

```
$ ssp 0x61001000 0x61080000 hello.exe
```

This will step the whole program. It will take at least 8 minutes on a PII/300 (yes, really). When it's done, it will create a file called "gmon.out". You can turn this data file into a readable report with **gprof**:

```
$ gprof -b cygwin1.dll
```

The "-b" means 'skip the help pages'. You can omit this until you're familiar with the report layout. The **gprof** documentation explains a lot about this report, but **ssp** changes a few things. For example, the first part of the report reports the amount of time spent in each function, like this:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 10.02    231.22    72.43       46  1574.57  1574.57  strcspn
  7.95    288.70    57.48      130   442.15   442.15  strncasematch
```

The "seconds" columns are really CPU opcodes, 1/100 second per opcode. So, "231.22" above means 23,122 opcodes. The ms/call values are 10x too big; 1574.57 means 157.457 opcodes per call. Similar adjustments need to be made for the "self" and "children" columns in the second part of the report.

OK, so now we've got a huge report that took a long time to generate, and we've identified a spot we want to work on optimizing. Let's say it's the time() function. We can use SSP to selectively profile this function by using OutputDebugString() to control SSP from within the program. Here's a sample program:

```
#include <windows.h>
main()
{
  time_t t;
  OutputDebugString("ssp on");
  time(&t);
  OutputDebugString("ssp off");
}
```

Then, add the `-d` option to ssp to default to *disabling* profiling. The program will run at full speed until the first OutputDebugString, then step until the second. You can then use **gprof** (as usual) to see the performance profile for just that portion of the program's execution.

There are many options to ssp. Since step-profiling makes your program run about 1,000 times slower than normal, it's best to understand all the options so that you can narrow down the parts of your program you need to single-step.

`-v` - verbose. This prints messages about threads starting and stopping, OutputDebugString calls, DLLs loading, etc.

`-t` and `-c` - tracing. With `-t`, *every* step's address is written to the file "trace.ssp". This can be used to help debug functions, since it can trace multiple threads. Clever use of scripts can match addresses with disassembled opcodes if needed. Warning: creates *huge* files, very quickly. `-c` prints each address to the console, useful for debugging key chunks of assembler. Use `addr2line -C -f -s -e foo.exe < trace.ssp > lines.ssp` and then `perl cvttrace` to convert to symbolic traces.

`-s` - subthreads. Usually, you only need to trace the main thread, but sometimes you need to trace all threads, so this enables that. It's also needed when you want to profile a function that only a subthread calls. However, using OutputDebugString automatically enables profiling on the thread that called it, not the main thread.

`-l` - dll profiling. Generates a pretty table of how much time was spent in each dll the program used. No sense optimizing a function in your program if most of the time is spent in the DLL. I usually use the `-v`, `-s`, and `-l` options:

```
$ ssp -v -s -l -d 0x61001000 0x61080000 hello.exe
```

3.7.14. strace

```
Usage: strace.exe [OPTIONS] <command-line>
Usage: strace.exe [OPTIONS] -p <pid>
Trace system calls and signals

  -b, --buffer-size=SIZE       set size of output file buffer
  -d, --no-delta               don't display the delta-t microsecond timestamp
  -f, --trace-children         trace child processes (toggle - default true)
  -h, --help                   output usage information and exit
  -m, --mask=MASK              set message filter mask
  -n, --crack-error-numbers    output descriptive text instead of error
                               numbers for Windows errors
  -o, --output=FILENAME        set output file to FILENAME
  -p, --pid=n                  attach to executing program with cygwin pid n
  -q, --quiet                  toggle "quiet" flag.  Defaults to on if "-p",
```

```
                              off otherwise.
-S, --flush-period=PERIOD    flush buffered strace output every PERIOD secs
-t, --timestamp              use an absolute hh:mm:ss timestamp insted of
                              the default microsecond timestamp.  Implies -d
-T, --toggle                 toggle tracing in a process already being
-u, --usecs                  toggle printing of microseconds timestamp
                              traced. Requires -p <pid>
-v, --version                output version information and exit
-w, --new-window             spawn program under test in a new window

  MASK can be any combination of the following mnemonics and/or hex values
  (0x is optional).  Combine masks with '+' or ',' like so:

                    --mask=wm+system,malloc+0x00800

  Mnemonic Hex     Corresponding Def  Description
  =========================================================================
  all      0x00001 (_STRACE_ALL)      All strace messages.
  flush    0x00002 (_STRACE_FLUSH)    Flush output buffer after each message.
  inherit  0x00004 (_STRACE_INHERIT)  Children inherit mask from parent.
  uhoh     0x00008 (_STRACE_UHOH)     Unusual or weird phenomenon.
  syscall  0x00010 (_STRACE_SYSCALL)  System calls.
  startup  0x00020 (_STRACE_STARTUP)  argc/envp printout at startup.
  debug    0x00040 (_STRACE_DEBUG)    Info to help debugging.
  paranoid 0x00080 (_STRACE_PARANOID) Paranoid info.
  termios  0x00100 (_STRACE_TERMIOS)  Info for debugging termios stuff.
  select   0x00200 (_STRACE_SELECT)   Info on ugly select internals.
  wm       0x00400 (_STRACE_WM)       Trace Windows msgs (enable _strace_wm).
  sigp     0x00800 (_STRACE_SIGP)     Trace signal and process handling.
  minimal  0x01000 (_STRACE_MINIMAL)  Very minimal strace output.
  exitdump 0x04000 (_STRACE_EXITDUMP) Dump strace cache on exit.
  system   0x08000 (_STRACE_SYSTEM)   Serious error; goes to console and log.
  nomutex  0x10000 (_STRACE_NOMUTEX)  Don't use mutex for synchronization.
  malloc   0x20000 (_STRACE_MALLOC)   Trace malloc calls.
  thread   0x40000 (_STRACE_THREAD)   Thread-locking calls.
```

The **strace** program executes a program, and optionally the children of the program, reporting any Cygwin DLL output from the program(s) to stdout, or to a file with the `-o` option. With the `-w` option, you can start an strace session in a new window, for example:

```
$ strace -o tracing_output -w sh -c 'while true; do echo "tracing..."; done' &
```

This is particularly useful for **strace** sessions that take a long time to complete.

Note that **strace** is a standalone Windows program and so does not rely on the Cygwin DLL itself (you can verify this with **cygcheck**). As a result it does not understand symlinks. This program is mainly useful for debugging the Cygwin DLL itself.

3.7.15. umount

```
Usage: umount.exe [OPTION] [<posixpath>]
Unmount filesystems

  -h, --help                    output usage information and exit
  -U, --remove-user-mounts      remove all user mounts
  -v, --version                 output version information and exit
```

The **umount** program removes mounts from the mount table in the current session. If you specify a POSIX path that corresponds to a current mount point, **umount** will remove it from the current mount table. Note that you can only remove user mount points. The `-U` flag may be used to specify removing all user mount points from the current user session.

See Section 3.1.2 for more information on the mount table.

3.8. Using Cygwin effectively with Windows

Cygwin is not a full operating system, and so must rely on Windows for accomplishing some tasks. For example, Cygwin provides a POSIX view of the Windows filesystem, but does not provide filesystem drivers of its own. Therefore part of using Cygwin effectively is learning to use Windows effectively. Many Windows utilities provide a good way to interact with Cygwin's predominately command-line environment. For example, **ipconfig.exe** provides information about network configuration, and **net.exe** views and configures network file and printer resources. Most of these tools support the `/?` switch to display usage information.

Unfortunately, no standard set of tools included with all versions of Windows exists. If you are unfamiliar with the tools available on your system, here is a general guide. Windows NT 4.0 has only a basic set of tools, which later versions of Windows expanded. Microsoft also provides free downloads for Windows NT 4.0 (the Resource Kit Support Tools), Windows 2000 (the Resource Kit Tools), and XP (the Windows Support Tools). Generally, the younger the Windows version, the more complete are the on-board tools. Additionally, many independent sites such as download.com (http://download.com), simtel.net (http://simtel.net), and Microsoft's own Sysinternals (http://technet.microsoft.com/en-us/sysinternals/default.aspx) provide quite useful command-line utilities, as far as they are not already provided by Cygwin. A few Windows tools, such as **find.exe**, **link.exe** and **sort.exe**, may conflict with the Cygwin versions make sure that you use the full path (**/usr/bin/find**) or that your Cygwin `bin` directory comes first in your PATH.

3.8.1. Pathnames

Windows programs do not understand POSIX pathnames, so any arguments that reference the filesystem must be in Windows (or DOS) format or translated. Cygwin provides the **cygpath** utility for converting between Windows and POSIX paths. A complete description of its options and examples of its usage are in Section 3.7.2, including a shell script for starting Windows Explorer in any directory. The same format works for most Windows programs, for example

```
notepad.exe "$(cygpath -aw "Desktop/Phone Numbers.txt")"
```

A few programs require a Windows-style, semicolon-delimited path list, which **cygpath** can translate from a POSIX path with the `-p` option. For example, a Java compilation from **bash** might look like this:

```
javac -cp "$(cygpath -pw "$CLASSPATH")" hello.java
```

Since using quoting and subshells is somewhat awkward, it is often preferable to use **cygpath** in shell scripts.

3.8.2. Console Programs

Another issue is receiving output from or giving input to console-based Windows programs. Unfortunately, interacting with Windows console applications is not a simple matter of using a translation utility. Windows console applications are designed to run under **cmd.exe**, and some do not deal gracefully with other situations. Cygwin can receive console input only if it is also running in a console window since Windows does not provide any way to attach to the backend of the console device. Another traditional Unix input/output method, ptys (pseudo-terminals), is supported by Cygwin but not entirely by Windows. The basic problem is that a Cygwin pty is a pipe and some Windows applications do not like having their input or output redirected to pipes.

To help deal with these issues, Cygwin supports customizable levels of Windows versus Unix compatibility behavior. To be most compatible with Windows programs, use a DOS prompt, running only the occasional Cygwin command or script. Next would be to run **bash** within a default DOS box. To make Cygwin more Unix compatible in this case, set CYGWIN=tty (see Section 3.5). Alternatively, the optional rxvt package provides a native-Windows version of the popular X11 terminal emulator (it is not necessary to set CYGWIN=tty with **rxvt**). Using **rxvt.exe** provides the most Unix-like environment, but expect some compatibility problems with Windows programs.

3.8.3. Cygwin and Windows Networking

Many popular Cygwin packages, such as ncftp, lynx, and wget, require a network connection. Since Cygwin relies on Windows for connectivity, if one of these tools is not working as expected you may need to troubleshoot using Windows tools. The first test is to see if you can reach the URL's host with **ping.exe**, one of the few utilities included with every Windows version since Windows 95. If you chose to install the inetutils package, you may have both Windows and Cygwin versions of utilities such as **ftp** and **telnet**. If you are having problems using one of these programs, see if the alternate one works as expected.

There are a variety of other programs available for specific situations. If your system does not have an always-on network connection, you may be interested in **rasdial.exe** for automating dialup connections. Users who frequently change their network configuration can script these changes with **netsh.exe** (Windows 2000 and later). For proxy users, the open source NTLM Authorization Proxy Server (http://apserver.sourceforge.net) or the no-charge Hummingbird SOCKS Proxy (http://www.hummingbird.com/products/nc/socks/index.html) may allow you to use Cygwin network programs in your environment.

3.8.4. The cygutils package

The optional cygutils package contains miscellaneous tools that are small enough to not require their own package. It is not included in a default Cygwin install; select it from the Utils category in **setup.exe**. Several of the cygutils tools are useful for interacting with Windows.

One of the hassles of Unix-Windows interoperability is the different line endings on text files. As mentioned in Section 3.2, Unix tools such as **tr** can convert between CRLF and LF endings, but cygutils provides several dedicated programs: **conv**, **d2u**, **dos2unix**, **u2d**, and **unix2dos**. Use the `--help` switch for usage information.

3.8.5. Creating shortcuts with cygutils

Another problem area is between Unix-style links, which link one file to another, and Microsoft .lnk files, which provide a shortcut to a file. They seem similar at first glance but, in reality, are fairly different. By default, Cygwin uses a mechanism that creates symbolic links that are compatible with standard Microsoft .lnk files. However, they do not include much of the information that is available in a standard Microsoft shortcut, such as the working directory, an icon, etc. The cygutils package includes a **mkshortcut** utility for creating standard Microsoft .lnk files.

If Cygwin handled these native shortcuts like any other symlink, you could not archive Microsoft .lnk files into **tar** archives and keep all the information in them. After unpacking, these shortcuts would have lost all the extra information and would be no different than standard Cygwin symlinks. Therefore these two types of links are treated differently. Unfortunately, this means that the usual Unix way of creating and using symlinks does not work with Windows shortcuts.

3.8.6. Printing with cygutils

There are several options for printing from Cygwin, including the **lpr** found in cygutils (not to be confused with the native Windows **lpr.exe**). The easiest way to use cygutils' **lpr** is to specify a default device name in the PRINTER environment variable. You may also specify a device on the command line with the `-d` or `-P` options, which will override the environment variable setting.

A device name may be a UNC path (`\\server_name\printer_name`), a reserved DOS device name (`prn`, `lpt1`), or a local port name that is mapped to a printer share. Note that forward slashes may be used in a UNC path (`//server_name/printer_name`), which is helpful when using **lpr** from a shell that uses the backslash as an escape character.

lpr sends raw data to the printer; no formatting is done. Many, but not all, printers accept plain text as input. If your printer supports PostScript, packages such as a2ps and enscript can prepare text files for printing. The ghostscript package also provides some translation from PostScript to various native printer languages. Additionally, a native Windows application for printing PostScript, **gsprint**, is available from the Ghostscript website (http://www.cs.wisc.edu/~ghost/).

Chapter 4. Programming with Cygwin

4.1. Using GCC with Cygwin

4.1.1. Console Mode Applications

Use gcc to compile, just like under UNIX. Refer to the GCC User's Guide for information on standard usage and options. Here's a simple example:

Example 4-1. Building Hello World with GCC

```
bash$ gcc hello.c -o hello.exe
bash$ hello.exe
Hello, World

bash$
```

4.1.2. GUI Mode Applications

Cygwin allows you to build programs with full access to the standard Windows 32-bit API, including the GUI functions as defined in any Microsoft or off-the-shelf publication. However, the process of building those applications is slightly different, as you'll be using the GNU tools instead of the Microsoft tools.

For the most part, your sources won't need to change at all. However, you should remove all __export attributes from functions and replace them like this:

```
int foo (int) __attribute__ ((__dllexport__));

int
foo (int i)
```

The Makefile is similar to any other UNIX-like Makefile, and like any other Cygwin makefile. The only difference is that you use **gcc -mwindows** to link your program into a GUI application instead of a command-line application. Here's an example:

```
myapp.exe : myapp.o myapp.res
 gcc -mwindows myapp.o myapp.res -o $@

myapp.res : myapp.rc resource.h
 windres $< -O coff -o $@
```

Note the use of `windres` to compile the Windows resources into a COFF-format `.res` file. That will include all the bitmaps, icons, and other resources you need, into one handy object file. Normally, if you omitted the "-O coff" it would create a Windows `.res` format file, but we can only link COFF objects. So, we tell `windres` to produce a COFF object, but for compatibility with the many examples that assume your linker can handle Windows resource files directly, we maintain the `.res` naming convention. For more information on `windres`, consult the Binutils manual.

The following is a simple GUI-mode "Hello, World!" program to help get you started:

```
/*-------------------------------------------------*/
/* hellogui.c - gui hello world                    */
/* build: gcc -mwindows hellogui.c -o hellogui.exe */
/*-------------------------------------------------*/
#include <windows.h>

char glpszText[1024];

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
  HINSTANCE hPrevInstance,
  LPSTR lpCmdLine,
  int nCmdShow)
{
 sprintf(glpszText,
  "Hello World\nGetCommandLine(): [%s]\n"
  "WinMain lpCmdLine: [%s]\n",
  lpCmdLine, GetCommandLine() );

 WNDCLASSEX wcex;

 wcex.cbSize = sizeof(wcex);
 wcex.style = CS_HREDRAW | CS_VREDRAW;
 wcex.lpfnWndProc = WndProc;
 wcex.cbClsExtra = 0;
 wcex.cbWndExtra = 0;
 wcex.hInstance = hInstance;
 wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
 wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
 wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
 wcex.lpszMenuName = NULL;
 wcex.lpszClassName = "HELLO";
 wcex.hIconSm = NULL;

 if (!RegisterClassEx(&wcex))
  return FALSE;

 HWND hWnd;
 hWnd = CreateWindow("HELLO", "Hello", WS_OVERLAPPEDWINDOW,
  CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, hInstance, NULL);

 if (!hWnd)
```

```
  return FALSE;

 ShowWindow(hWnd, nCmdShow);
 UpdateWindow(hWnd);

 MSG msg;
 while (GetMessage(&msg, NULL, 0, 0))
 {
  TranslateMessage(&msg);
  DispatchMessage(&msg);
 }

 return msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
 PAINTSTRUCT ps;
 HDC hdc;

 switch (message)
 {
  case WM_PAINT:
   hdc = BeginPaint(hWnd, &ps);
   RECT rt;
   GetClientRect(hWnd, &rt);
   DrawText(hdc, glpszText, strlen(glpszText), &rt, DT_TOP | DT_LEFT);
   EndPaint(hWnd, &ps);
   break;
  case WM_DESTROY:
   PostQuitMessage(0);
   break;
  default:
   return DefWindowProc(hWnd, message, wParam, lParam);
 }
 return 0;
}
```

4.2. Debugging Cygwin Programs

When your program doesn't work right, it usually has a "bug" in it, meaning there's something wrong with the program itself that is causing unexpected results or crashes. Diagnosing these bugs and fixing them is made easy by special tools called *debuggers*. In the case of Cygwin, the debugger is GDB, which stands for "GNU DeBugger". This tool lets you run your program in a controlled environment where you can investigate the state of your program while it is running or after it crashes. Crashing programs sometimes create "core" files. In Cygwin these are regular text files that cannot be used directly by GDB.

Before you can debug your program, you need to prepare your program for debugging. What you need to do is add `-g` to all the other flags you use when compiling your sources to objects.

Example 4-2. Compiling with -g

```
bash$ gcc -g -O2 -c myapp.c
bash$ gcc -g myapp.c -o myapp
```

What this does is add extra information to the objects (they get much bigger too) that tell the debugger about line numbers, variable names, and other useful things. These extra symbols and debugging information give your program enough information about the original sources so that the debugger can make debugging much easier for you.

To invoke GDB, simply type **gdb myapp.exe** at the command prompt. It will display some text telling you about itself, then `(gdb)` will appear to prompt you to enter commands. Whenever you see this prompt, it means that gdb is waiting for you to type in a command, like **run** or **help**. Oh `:-)` type **help** to get help on the commands you can type in, or read the [GDB User's Manual] for a complete description of GDB and how to use it.

If your program crashes and you're trying to figure out why it crashed, the best thing to do is type **run** and let your program run. After it crashes, you can type **where** to find out where it crashed, or **info locals** to see the values of all the local variables. There's also a **print** that lets you look at individual variables or what pointers point to.

If your program is doing something unexpected, you can use the **break** command to tell gdb to stop your program when it gets to a specific function or line number:

Example 4-3. "break" in gdb

```
(gdb) break my_function
(gdb) break 47
```

Now, when you type **run** your program will stop at that "breakpoint" and you can use the other gdb commands to look at the state of your program at that point, modify variables, and **step** through your program's statements one at a time.

Note that you may specify additional arguments to the **run** command to provide command-line arguments to your program. These two cases are the same as far as your program is concerned:

Example 4-4. Debugging with command line arguments

```
bash$ myprog -t foo --queue 47

bash$ gdb myprog
(gdb) run -t foo --queue 47
```

4.3. Building and Using DLLs

DLLs are Dynamic Link Libraries, which means that they're linked into your program at run time instead of build time. There are three parts to a DLL:

- the exports
- the code and data
- the import library

The code and data are the parts you write - functions, variables, etc. All these are merged together, like if you were building one big object files, and put into the dll. They are not put into your .exe at all.

The exports contains a list of functions and variables that the dll makes available to other programs. Think of this as the list of "global" symbols, the rest being hidden. Normally, you'd create this list by hand with a text editor, but it's possible to do it automatically from the list of functions in your code. The `dlltool` program creates the exports section of the dll from your text file of exported symbols.

The import library is a regular UNIX-like `.a` library, but it only contains the tiny bit of information needed to tell the OS how your program interacts with ("imports") the dll. This information is linked into your `.exe`. This is also generated by `dlltool`.

4.3.1. Building DLLs

This page gives only a few simple examples of gcc's DLL-building capabilities. To begin an exploration of the many additional options, see the gcc documentation and website, currently at http://gcc.gnu.org/

Let's go through a simple example of how to build a dll. For this example, we'll use a single file `myprog.c` for the program (`myprog.exe`) and a single file `mydll.c` for the contents of the dll (`mydll.dll`).

Fortunately, with the latest gcc and binutils the process for building a dll is now pretty simple. Say you want to build this minimal function in mydll.c:

```
#include <stdio.h>

int
hello()
{
  printf ("Hello World!\n");
}
```

First compile mydll.c to object code:

```
gcc -c mydll.c
```

Then, tell gcc that it is building a shared library:

```
gcc -shared -o mydll.dll mydll.o
```

That's it! To finish up the example, you can now link to the dll with a simple program:

```
int
main ()
{
  hello ();
}
```

Then link to your dll with a command like:

```
gcc -o myprog myprog.c -L./ -lmydll
```

However, if you are building a dll as an export library, you will probably want to use the complete syntax:

```
gcc -shared -o cyg${module}.dll \
    -Wl,--out-implib=lib${module}.dll.a \
    -Wl,--export-all-symbols \
    -Wl,--enable-auto-import \
    -Wl,--whole-archive ${old_libs} \
    -Wl,--no-whole-archive ${dependency_libs}
```

The name of your library is `${module}`, prefixed with `cyg` for the DLL and `lib` for the import library. Cygwin DLLs use the `cyg` prefix to differentiate them from native-Windows MinGW DLLs, see the MinGW website (http://mingw.org) for more details. `${old_libs}` are all your object files, bundled together in static libs or single object files and the `${dependency_libs}` are import libs you need to link against, e.g **`'-lpng -lz -L/usr/local/special -lmyspeciallib'`**.

4.3.2. Linking Against DLLs

If you have an existing DLL already, you need to build a Cygwin-compatible import library. If you have the source to compile the DLL, see Section 4.3.1 for details on having `gcc` build one for you. If you do not have the source or a supplied working import library, you can get most of the way by creating a .def file with these commands (you might need to do this in `bash` for the quoting to work correctly):

```
echo EXPORTS > foo.def
nm foo.dll | grep ' T _' | sed 's/.* T _//' >> foo.def
```

Note that this will only work if the DLL is not stripped. Otherwise you will get an error message: "No symbols in foo.dll".

Once you have the `.def` file, you can create an import library from it like this:

```
dlltool --def foo.def --dllname foo.dll --output-lib foo.a
```

4.4. Defining Windows Resources

`windres` reads a Windows resource file (`*.rc`) and converts it to a res or coff file. The syntax and semantics of the input file are the same as for any other resource compiler, so please refer to any publication describing the Windows resource format for details. Also, the `windres` program itself is fully documented in the Binutils manual. Here's an example of using it in a project:

```
myapp.exe : myapp.o myapp.res
 gcc -mwindows myapp.o myapp.res -o $@

myapp.res : myapp.rc resource.h
 windres $< -O coff -o $@
```

What follows is a quick-reference to the syntax `windres` supports.

```
id ACCELERATORS suboptions
BEG
"^C" 12
"Q" 12
65 12
65 12 , VIRTKEY ASCII NOINVERT SHIFT CONTROL ALT
65 12 , VIRTKEY, ASCII, NOINVERT, SHIFT, CONTROL, ALT
(12 is an acc_id)
END

SHIFT, CONTROL, ALT require VIRTKEY


id BITMAP memflags "filename"
memflags defaults to MOVEABLE


id CURSOR memflags "filename"
memflags defaults to MOVEABLE,DISCARDABLE


id DIALOG memflags exstyle x,y,width,height styles BEG controls END
id DIALOGEX memflags exstyle x,y,width,height styles BEG controls END
id DIALOGEX memflags exstyle x,y,width,height,helpid styles BEG controls END

memflags defaults to MOVEABLE
exstyle may be EXSTYLE=number
styles: CAPTION "string"
```

```
 CLASS id
 STYLE  FOO | NOT FOO | (12)
 EXSTYLE number
 FONT number, "name"
 FONT number, "name",weight,italic
 MENU id
 CHARACTERISTICS number
 LANGUAGE number,number
 VERSIONK number
controls:
 AUTO3STATE params
 AUTOCHECKBOX params
 AUTORADIOBUTTON params
 BEDIT params
 CHECKBOX params
 COMBOBOX params
 CONTROL ["name",] id, class, style, x,y,w,h [,exstyle] [data]
 CONTROL ["name",] id, class, style, x,y,w,h, exstyle, helpid [data]
 CTEXT params
 DEFPUSHBUTTON params
 EDITTEXT params
 GROUPBOX params
 HEDIT params
 ICON ["name",] id, x,y [data]
 ICON ["name",] id, x,y,w,h, style, exstyle [data]
 ICON ["name",] id, x,y,w,h, style, exstyle, helpid [data]
 IEDIT params
 LISTBOX params
 LTEXT params
 PUSHBOX params
 PUSHBUTTON params
 RADIOBUTTON params
 RTEXT params
 SCROLLBAR params
 STATE3 params
 USERBUTTON "string", id, x,y,w,h, style, exstyle
params:
 ["name",] id, x, y, w, h, [data]
 ["name",] id, x, y, w, h, style [,exstyle] [data]
 ["name",] id, x, y, w, h, style, exstyle, helpid [data]

[data] is optional BEG (string|number) [,(string|number)] (etc) END


id FONT memflags "filename"
memflags defaults to MOVEABLE|DISCARDABLE

id ICON memflags "filename"
memflags defaults to MOVEABLE|DISCARDABLE

LANGUAGE num,num

id MENU options BEG items END
```

```
items:
 "string", id, flags
 SEPARATOR
 POPUP "string" flags BEG menuitems END
flags:
 CHECKED
 GRAYED
 HELP
 INACTIVE
 MENUBARBREAK
 MENUBREAK

id MENUEX suboptions BEG items END
items:
 MENUITEM "string"
 MENUITEM "string", id
 MENUITEM "string", id, type [,state]
 POPUP "string" BEG items END
 POPUP "string", id BEG items END
 POPUP "string", id, type BEG items END
 POPUP "string", id, type, state [,helpid] BEG items END

id MESSAGETABLE memflags "filename"
memflags defaults to MOVEABLE

id RCDATA suboptions BEG (string|number) [,(string|number)] (etc) END

STRINGTABLE suboptions BEG strings END
strings:
 id "string"
 id, "string"

(User data)
id id suboptions BEG (string|number) [,(string|number)] (etc) END

id VERSIONINFO stuffs BEG verblocks END
stuffs: FILEVERSION num,num,num,num
 PRODUCTVERSION num,num,num,num
 FILEFLAGSMASK num
 FILEOS num
 FILETYPE num
 FILESUBTYPE num
verblocks:
 BLOCK "StringFileInfo" BEG BLOCK BEG vervals END END
 BLOCK "VarFileInfo" BEG BLOCK BEG vertrans END END
vervals: VALUE "foo","bar"
vertrans: VALUE num,num



suboptions:
 memflags
 CHARACTERISTICS num
```

```
 LANGUAGE num,num
 VERSIONK num

memflags are MOVEABLE/FIXED PURE/IMPURE PRELOAD/LOADONCALL DISCARDABLE
```