Frequently Asked Questions (FAQ) for Circle DikuMud with Answers

Alex Fletcher, furry@circlemud.org Revision 2.6, 28 August 1998

This file is intended to cover common questions related to the CircleMUD distributon source by *Jeremy Elson* <http://www.cs.jhu.edu/jelson/> and not general DikuMud questions. Any contributions and corrections are more than welcome. It is currently maintained by *Alex Fletcher* <mailto:furry@circlemud.org> (aka Furry). Please direct corrections to this address. The original author was Ryan Watkins (aka VampLestat). More information about CircleMUD, including up-to-date versions of this documentation in ASCII and Postscript, can be found at the *CircleMUD Home Page* <http://www.circlemud.org/> or *FTP site* <ftp://ftp.circlemud.org/pub/CircleMUD/>.

Contents

1 Introduction

1.1 I've never played a MUD before. What should I do?

Don't try to use your own copy of CircleMUD! There are two levels of MUD users: players and administrators. Administrators do what you're trying to do now – get a copy of a MUD's source code, compile it, and run it. Players use MUDs that are being administered by someone else. If you try to actually run a MUD before you've ever played one, you'll get very confused indeed! Your best bet for now is to play someone else's MUD first. There are a large number of excellent MUDs out there already, some of which are based on CircleMUD code. A good place to start looking is the *Mud Connector* `<http://www.mudconnect.com/>` The CircleMUD web server also has a smaller *CircleMUD Site List* `<http://cambot.circlemud.org/sites.html>`.

1.2 I'm new to C and/or coding. What do I do?

First, a Mud is not a learning project. It has thousands of lines to it many of which are obscure and unclear to even moderately skilled programmers. Those little, "Hello, world," programs are for learning, maybe little math tests, etc. A Mud is a pretty ambitous project to start with. That's like trying to run before you can walk, and while there's more difficult things than a mud to start with, there's a ton of easier things you should start with. Second, if you are persistent, get a good C reference book and read the code, try to comprehend what everything is doing (to some small extent). You should probably avoid comm.c as it is home to the socket functions which general C books don't cover and are usually explained in other books on network programming. Then try small projects, something similar to what already exists. This way, you can get away with a cut-and-paste job and changing some of the code. Adding a simple version of races isn't all that difficult, just examine the class code in class.c and the CON_QCLASS block in interpreter.c, cut, paste, and modify. Also look at `structs.h`, for `"#define CLASS_"` You'll begin understanding more and more of the code as you copy and change it, eventually you'll be able to write a whole function by yourself. Spend time learning, going with haste will hurt you more than it will help you.

1.3 I want to build my own mud. Where do I start?

Many common questions arise from new mud admins. It is a good idea to pay attention to them and take their answers to heart. These include things like the following:

I don't have any coding experience with MUDs, but do have a lot of ideas for my own. I have played many MUDs for a LONG time, though.

Read the FAQ. Mud Experience doesn't help a huge amount. Code experience does.

I am interested in having a level system of 1-50 mortal and 51-60 imms. I am also interested in adding races and classes. How can I accomplish these things?

By checking the FTP Site under contrib. Learn a lot from there. Learn from the Snippets page, and *know* the FAQ.

Also, is there anything that I should know about CircleMUD being a "newbie" to it?

See the above comment.

1.4 What is CircleMUD?

CircleMUD is a DikuMud derivitave, developed by *Jeremy Elson* `<mailto:jelson@circlemud.org>` and is from the Gamma v0.0 of DikuMud created by Hans Henrik Staerfeldt, Katja Nyboe, Tom Madsen, Michael Seifert and Sebastian Hammer at DIKU (Computer Science Instutute at Copenhagen University). Note that CircleMUD is a Diku derivative, so its users must follow the DIKU license agreement—most notably that it cannot be used to make money in *ANY* way, the original developers' names must be in the login screen that the `credits` command always presents the same information, etc.

Quoting from CircleMUD's `release.doc`:

```
"CircleMUD is highly developed from the programming side, but highly UNdeveloped on
the game-playing side.  So, if you're looking for a huge MUD with billions of spells,
skills, classes, races, and areas, Circle will probably disappoint you severely.  Circle
still has only the 4 original Diku classes, the original spells, the original skills,
```

```
and about a couple dozen areas.  On the other hand, if you're looking for a highly
stable, well-developed, well-organized "blank slate" MUD on which you can put your OWN
ideas for spells, skills, classes, and areas, then Circle might be just what you're
looking for."
```

The latest full release of Circle is 2.20, released on November 17, 1993. Currently 3.0 is in beta, at patch level 14, released on July 3, 1998.

1.5 What is the history of CircleMUD?

- Version 2.20: November 17, 1993
- Version 2.11: September 19, 1993
- Version 2.10: September 1, 1993
- Version 2.02: Early August
- Version 2.01: July 20, 1993
- Version 2.00: July 16, 1993

Version 3.00 is currently in beta right now and is up to patchlevel 14. The final release is due out Real Soon Now(tm). Don't bother posting requests for it. It will be out when Jeremy gets it done, and it will be announced on the mailing list and the newsgroup rec.games.mud.diku.

1.6 Where is the original CircleMUD so I can check it out?

CircleMUD is a public code base, freely distributable, but the author of Circle doesn't actually run one personally. There used to be CircleMUD, and while Jeremy Elson continues to develop it, there is no original CircleMUD any more. To see other MUDs that are using the CircleMUD code base, check out this *Site List* `<http://cambot.circlemud.org/sites.html>`.

1.7 What is UNIX?

UNIX is not an operating system of itself, it's a type (flavour, if you will) of operating systems. Many different kinds of UNIXes exist. Some of them are free, some of them are not. How to tell if you have a UNIX operating system? Well, UNIXes have the 'ps' command, tend to have a '%' or '#' prompt, give you a home directory, 'who' will show who else is on the system, etc. Many UNIX systems (such as Linux) strive to be POSIX compatible, so you'll probably see POSIX mentioned, too. POSIX is, roughly, the standards which UNIX operating systems go by. It says what makes an operating system part of the UNIX family and so forth. Some UNIX operating systems are not 100% POSIX compatible, actually, most aren't. The following are types of UNIX (but not all the existing flavours): Linux, FreeBSD, BSD, BSDi, Solaris. There are others. UNIX operating systems are command-based and Microsoft does not make a variant.

2 Resources

2.1 Where do I find the source code for CircleMUD?

Circle's complete source code and areas are available for anonymous FTP at the CircleMUD *FTP site* <ftp://ftp.circlemud.org/pub/CircleMUD> There is also a CircleMUD *Home Page* <http://www.circlemud.org/> for CircleMUD information.

2.2 Where do I find areas, etc. for CircleMUD?

A number of CircleMUD based Implementors have submitted areas to the public and they are *archived* <ftp://ftp.circlemud.org/pub/CircleMUD/contrib/areas/> at the same sites as the CircleMUD source. You can also use the *ftp2 mirror* <ftp://ftp2.circlemud.org/pub/CircleMUD/contrib/> or *van mirror site.* <ftp://ftp.van.ml.org/pub/CircleMUD/>. If you cannot use FTP, you can contact one of the site maintainers and request a file. The maintainers are *Alex Fletcher* <mailto:furry@circlemud.org>, *Chris Epler* <mailto:cepler@circlemud.org>, and *George Greer* <mailto:greerga@circlemud.org>, If the FTP site is down, please contact Alex at his *alternate address* <mailto:fletcher@democracy.queensu.ca> since he will be able to find out the situation. There is also a *code snippets site* <http://mouse.van.ml.org/> where a number of code suggestions and hints are located.

2.3 I have questions about CircleMUD. Where should I go?

If you have general questions about the MUD such as how to get it running, how to add new spells, how to add new skills, etc., the first place you should look is the documentation. 'coding.doc' will have information about how to add new spells, skills, commands, etc. 'building.doc' has information about how to create new worlds, how to read the database files, etc. There are many other documents in the doc directory with useful information.

There is also a new project, started in June of 1996, called the *CircleMUD Documentation Project* <http://www.circlemud.org/jelson/circle/cdp/> which will eventually be a repository for all Circle-related information. It's still being built as of this writing, but hopefully will become a valuable resource as more documentation is added.

If you still have questions *after* reading the doucmentation, you can try asking on the CircleMUD mailing list (see next section).

If you have a question you think is too "newbie" for the mailing list, try the *help database* <mailto:help@circlemud.org>. Previous questions are accessible though the *bugs database* <http://bugs.circlemud.org/>. That is also, incidently, where you should go to report bugs.

You can also contact the author, *Jeremy Elson* <mailto:jelson@circlemud.org>. *George Greer* <mailto:greerga@circlemud.org> can also be contacted for most problems.

2.4 How do I subscribe/unsubscribe to the Mailing List?

There is a CircleMUD mailing list for coders, builders, and administrators. To subscribe, send a message to the *list server* <mailto:listserv@post.queensu.ca> with a message body of subscribe circle <first name> <last name>. To unsubscribe from the list send a message to <listserv@post.queensu.ca> with the words `unsubscribe circle` as the message body. *DO NOT* send subscription or unsubscription requests to the list in general. There are hundreds of people on the list, and it will only irritate a ton of people who have no power to remove you from the list. Read the *Mailing List FAQ* `<http://democracy.queensu.ca/fletcher/Circle/list_faq.html>` for more information.

2.5 To what platforms has CircleMUD been ported?

Version 3.0, although still officially in beta-testing, is very portable because it uses the GNU autoconf system, meaning you only need to type "configure" to have it automatically determine various features of your system and configure the code accordingly. 3.0 compiles without changes under most BSD and SVR4 systems, including SunOS, Solaris, Ultrix, IRIX, AIX, Linux, BSD/OS, HP/UX, and others.

Version 3.0 is also being ported to various non-UNIX platforms. As of patchlevel 14, you can compile Circle under OS/2 2.x and 3.x with the OS/2 port of gcc, Windows 95/NT using Microsoft Visual C++ version 4.0 or 5.0, Borland (now Inprise) C++ 4.5, Watcom v.11, Cygnus GNU-WIN32, LCC, Macintosh with CodeWarrior, Amiga, and Acorn RiscOS.

The older version of the code, Version 2.20, compiles mainly on BSD UNIX systems but has trouble under SVR4-based systems such as Solaris. The author has personally compiled and tested v2.20 under Ultrix 4.0, IRIX 4.0.1, 4.0.4 and 4.0.5, SunOS 4.1.1 and 4.1.3, AIX 3.2, Linux 0.99.x and 1.0.x, and ConvexOS V10.2. Users have reported that v2.20 compiles with relatively minor changes under NeXTStep 2.1 and 3.0, and HP/UX 9.0.

Jean-Jack Riethoven `<mailto:J.J.M.Riethoven@ab.agro.nl>` ported CircleMUD version 2.20 to the Amiga and has contributed his code for version 3.0 of CircleMUD. Questions about the Amiga source should be directed to Jean-Jack Riethoven, not Jeremy Elson or George Greer.

2.6 How can I submit code or areas for use with CircleMUD?

There is a special *uploads area* `<ftp://upload.circlemud.org/pub/CircleMUD/incoming>` in the CircleMUD domain for submissions of code, areas, utilities, scripts, and anything else that might be of use to Circle users. Please make sure only to upload to `upload.circlemud.org` and not any of the CircleMUD mirrors. These portions of code or areas will probably not be added to the full release of CircleMUD unless you make specific arrangements with Jeremy for code items, or with Alex for area/lib files.

2.7 How do I use a patch file and how can I make one?

Patch files are created and used using the "diff" and "patch" utilities, respectively. They can both be downloaded from the *GNU FTP Site* `<ftp://ftp.gnu.org/pub/gnu>` under the name "diffutils-xxx.tar.gz". If you happen to be using a DOS or Windows machine, you should get *patch for DOS* `<ftp://204.119.24.14/pub/patch/patch12.zip>`. This does not support long filenames (but can be used with short filenames). There is also a set of Windows 95 Utilities now available from the *Cygnus Gnu-Win32 Project* `<http://www.cygnus.com/misc/gnu-win32/>`.

These are the various parameters to use with diff (all work in general on unix based systems, but check out the help entries to be certain.

diff -uN [original_src_directory] [altered_src_directory] > Patch

`-u` is the unified output. ie. it tells diff to output the text what is called "patch" style. On some systems, you will have to use `-c` but it generates much larger and harder to follow patches.

`-N` Tells diff to treat files that are in one directory and not there in the other as being empty in the one they are not there. It allows entire files to be included into the patch.

`-r` recursive, add r to the uN above if you want it to recursively add in any subdirectories. (be careful with this one)

`-p` Tells diff to indicate what function is being "patched" in each section. This may not be supported by all versions of "diff."

If you download a patch file and would like to add it to your code, first make sure to read any instructions that the patch author might have written. The command used to add the patch may vary depending on how the patch was created. This should given in the first line of the patch or in the instructions. Normally, if using GNU patch with a unified diff, the command should be:

```
patch -u < [patchfile]
```

If the patch was created with a SYSV patcher (i.e. not a unified diff), the patch should be added with:

```
patch -c < [patchfile]
```

Of course, if the instructions state otherwise, ignore any instructions given here and follow the instructions given with the patchfile instead.

Finally, in modern patches, there are three characters of interest to note:

- ! :: The line changes between new and old.
- + :: This line is added to the old to make the new.
- - :: This line is removed from the old to make the new.
- The rest of the lines are just there to give you an idea of where to change.

3 Compiling CircleMUD

3.1 Why do I get many syntax errors with Sun's "cc" compiler?

Because Circle is written in ANSI C, and Sun's standard cc compiler isn't capable of compiling ANSI C code. You can try acc, Sun's ANSI C compiler, but it costs extra money to get it from Sun so your sysadmin may not have installed it. Most don't. The best solution is to get the GCC compiler from the GNU *FTP site* `<ftp://ftp.gnu.org/pub/gnu>` and install it, if you have enough time and space.

3.2 Why do I get all sorts of errors with "crypt" functions and header files?

(This information applies ONLY to Version 3.0 of the code.) CircleMUD normally uses the UNIX crypt() function to enrypt players' passwords. Because of export restrictions imposed by the U.S., some systems do not have the crypt() function. "configure" will usually be able to figure out whether or not your system

has crypt(), but if it guesses incorrectly and you see problems with the crypt() function or headers, you can manually disable password encryption by going into the sysdep.h source file and uncommenting the line that reads:

```
#define NOCRYPT
```

Be warned, however, that doing this causes the MUD to store players' passwords in plaintext rather than as encrypted strings. Also, if you move from a system which has crypt to one that doesn't, players won't be able to log in with their old passwords!

3.3 When I try to compile, why do I get a lot of undefined symbols referenced in comm.o for functions like socket, accept, and bind?

SVR4 systems require the socket and nsl libraries for network programs. You shouldn't see this error any more with version 3.0 because "configure" should automatically use those libraries for you; however, if you still have problems, try adding "-lsocket -lnsl" to the line in the Makefile that links all the object files together into the 'circle' binary.

If you're using V2.20 and you have this error, the best thing to do is simply to use V3.0 instead. If you insist on using 2.20, go into the Makefile and search for the comment next to "SVR4".

3.4 Every time I try to compile Circle (or any other piece of software) under Linux, it gives me errors and says it cannot find include header files in the `linux/` and `asm/` directories. What can I do?

Under Linux, you cannot compile any program unless you install the kernel source code because the kernel source includes the ANSI C header files. You need the files in `/usr/include/linux`, which are distributed separately from the rest of `/usr/include`.

The easiest way to do this if you're using the Slackware Linux distribution is simply to install the 'K' series of disks which is the kernel source.

Otherwise, you'll have to set up your include files manually. The easiest way to get these is to download kernel source from *ftp.kernel.org* `<ftp://ftp.kernel.org:/pub/linux/kernel>`. Get the kernel source that matches the kernel you're running (type 'uname -a' to find your kernel version). Then unpack the kernel into the `/usr/src` directory. It's 6 megabytes compressed, 25 megabytes uncompressed.

Read the README file that comes with the kernel, and make the symbolic links you need for `/usr/include/asm` and `/usr/include/linux`.

Now compile the MUD. This will take care of most of the errors. You may have to do 'make config' and 'make dep' in `/usr/src/linux` as well, in order to make `linux/config.h` and other files that get generated by these steps.

You can remove the whole kernel source tree except for `include/` at this point and get most of your 25 megs back.

(Thanks to Michael Chastain for providing this answer.)

3.5 I'm getting compilation errors from a header file, and I didn't even change it?

Okay, if you really didn't change "structs.h" then the error isn't in "structs.h". I think I've seen 2 cases where this has happened, the first, is that the header file you included right before the header file messing has an error in it. I can't really say much beyond that, but look for a missing semicolon, are any other errors you can find.

If you include files out of order, it can mess things up. For example, B.h has stuff in it that is defined in A.h, and if you include B.h before A.h, you can get errors, your best bet here is to mess with the order of the headers, making sure you put "conf.h" and "sysdep.h" at the top, followed by "structs.h", "utils.h", etc. Any file specific headers should be the last one included just for coding style.

3.6 I'm trying to compile the mud on Windows '95 and am having problems, what can I do?

The first thing to do is to make sure you are compiling a recent version of the source code. Patch Level 11 and onwards all support Windows '95 winsock sockets now. Second, you should ensure that you have carefully read the `README.WIN` file for instructions on what to include. Next, ensure that you are using a C compiler that supports long filenames (for example, MSVC 4.0 does, MSVC 1.0 does not). If you happen to be trying to patch something into your code, you should use *patch for DOS* `<ftp://204.119.24.14/pub/patch/patch12.zip>`. This does not support long filenames (but can be used with short filenames).

3.7 How can I do a "grep" on Windows 95?

1. Select "start menu"->"find"->"files or folders"
2. Enter the files/dirs to search in.
3. Select "Advanced"
4. In the "Containing Text" input box, type in the text you want.
5. Double click on a match to bring up the file that matched.

Even better is to use MSVC's find command (if you have it).

3.8 While compiling the mud, why do I get errors like "foo.c:1231: Undefined symbol '_whereamI' referenced from text segment"

You forgot to include a source file into the make. Go edit your Makefile and make sure all the necessary *.c files are in there, in particular, whichever C file defines the function that the compiler is complaining is undefined. If all else fails, try deleting all the *.o files and recompiling from scratch.

3.9 What is a parse error and how do I fix it?

A parsing error is often a missing or extra semicolon, parenthesis, or bracket ({).

If the parse error is before a semicolon at the end of a line of code, it is something on that line.

If it is at the beginning of a line within a function, it is usually a missing semicolon on the previous line.

If it is at the beginning of a function, count your brackets (especially the {} ones) in the previous function.

I can't think of any other parse errors. These are the ones I commonly see. With a bit of practice, they are very easy to locate and fix. For a more detailed explanation, check out the C Language FAQ.

3.10 I have this piece of code that calls bcopy(), bzero(), and bcmp() and it won't compile, so what can I do?

All three of these functions are fairly standard on BSD systems. However, they are not considered to be very portable, and thus should be redefined. For example, the equivalents for SYSV are:

```
#define bcopy(from,to,len)      memmove(to,from,len)
#define bzero(mem,len)          memset(mem,0,len)
#define bcmp(a,b,len)           memcmp(a,b,len)
```

3.11 My compiler doesn't have "strdup()", what can I do?

Use Circle's built-in `str_dup()` function instead.

3.12 I am having trouble with my "makefile", what could be the problem?

If you used cut and paste to insert items into your makefile, it is likely that you accidentally put spaces at the beginning of lines where tabs are needed. This is how the makefile must be constructed:

```
foo.o: foo.c conf.h sysdep.h structs.h utils.h interpreter.h \
  handler.h db.h
{TAB}$(CC) -c $(CFLAGS)
```

To add these lines properly, you can use `gcc` to assist you with the following shell script (from Daniel Koepke):

```
#!/bin/sh
gcc -MM $1 >> Makefile
echo "{TAB}\$(CC) -c \$(CFLAGS) $1" >> Makefile
```

To use this script, replace {TAB} with a tab, and then run the script like: `add_file foo.c`

3.13 How can I handle directories in C?

`Note that this seems only to be valid for UNIX OSes.` Handling of directories is accomplished through the `dirent.h` and `sys/types.h` files. The function `opendir()` returns a "DIR*" pointer (it's like `but not the same as` the "FILE *" pointer) when you pass it the name of a directory to open or NULL if it can't open the dir. After the directory has been opened, you can step through the files or search for particular files, etc. using `readdir()`, `seekdir()`, and `scandir()`. When you reach the end of the directory list, you can either go back to the start with `rewinddir()` or close the directory with `closedir()`. The following code (which has not been tested) should open a directory and go through it one by one and prints the filenames:

```
struct dirent * ffile;
DIR * my_dir;

if (!(my_dir = opendir("foo")))
  return;

while (1) {
  if (!(dirent = readdir(my_dir)))
    break;
  printf("%s\n", dirent->d_name);
}

closedir(my_dir);
```

The dirent structure contains only two useful elements, the file's name (`d_name`) and the files length (`d_reclen`).

Thanks to Daniel Koepke for the above.

4 Running CircleMUD

4.1 I typed "autorun" but then my terminal just froze.

autorun is a script which automatically runs, logs, and reboots the game for long-term runs. You should run autorun in the background by typing "./autorun &" – the MUD will start running in the background and you'll get the normal UNIX prompt back immediately (see section 4.3). The game will then run unattended until you explicitly shut it down.

On some systems, you may need to prepend "nohup" to the autorun command since some systems will kill off any processes left running when you leave the shell.

4.2 I typed "bin/circle" and got lots of boot messages, but then it said "Entering game loop" and froze.

It is not frozen, it is just waiting for people to connect. You have to run the MUD in the background by typing "bin/circle &" and then use telnet to connect to the game (see next section).

4.3 Okay, I think the MUD is running but why don't I get a login prompt?

In order to play the MUD, you must connect to it using the telnet command, i.e. "telnet localhost 4000".

4.4 How come I get this error when running my mud: "Error reading board: No such file or directory"

This is not a bad thing, all it means is that you have some boards on the mud and that it can't find the file for them. Since it can't find the file, the mud will just create the file on the fly and use that, so the next time something is posted to the board, the files will exist. However, if you did have files for the boards and you are suddenly getting this error, it means that the board files have been deleted or something similar.

4.5 I just got this SIGPIPE, what is it and what Can I Do About It?

Often it appears that other people send your system SIGPIPEs when their connection is closed, in fact, it is not the person sending the SIGPIPE, it is your system. The SIGPIPE is generated when your program attempts to write to descriptor which has no one listening to it. This occurs if the character is sent a message by the mud after connecting, but before the socket is flagged with an exception or reads 0 bytes. By default, CircleMUD ignores these SIGPIPEs, with the line `my_signal(SIGPIPE, SIG_IGN)` in `signal_setup()`. Where most people see the problems with SIGPIPE is while debugging with GDB. By default, GDB responds to a SIGPIPE by stoping the program, printing that a SIGPIPE was received, and passing it to the program. You can change the action taken by GDB by using the `'handle'` command. To stop the program from stoping at SIGPIPE, you would give GDB the command `'handle SIGPIPE nostop'`

4.6 When I run Circle under Linux, it tells me "gethostbyaddr: connection refused" when the MUD boots, and then dies. Why?

You need to make sure you have Networking and TCP/IP support compiled into your Linux kernel, even if you aren't actually connected to the Internet. The easiest way to do this if you're using Slackware is to install one of Slackware's precompiled networking kernels. Also, make sure to install Slackware's 'N' series of disks which contains other networking support files.

If Slackware's precompiled kernel isn't available you'll have to compile the kernel yourself. First make sure the kernel source is installed in `/usr/src/linux` (see section 3.1.4) and follow the instructions in `/usr/src/linux/README`. When you do 'make config' it will ask you a series of questions about which kernel features you want; make sure to answer "Y" to "Networking support" and "TCP/IP support".

4.7 When I run Circle under Windows 95, it tells me "Winsock error #10047" when the MUD boots, and then dies. Why?

You need to configure TCP/IP networking from the Network Control Panel, even if you are not connected to the Internet. From the Network Control Panel, select "Add Protocol", and under the vendor "Microsoft", choose "TCP/IP". It may ask you to insert the Windows 95 CDROM in order to copy the drivers onto your hard drive.

4.8 When I run Circle under Windows 95, players can't rent—their equipment is just dropped on the ground, syslogs don't work, so what is the problem?

The reason that objects aren't saved when your players quit is that certain unzip programs are buggy and don't completely recreate the MUD's directory structure (in particular, it doesn't create directories which have no files in them.) This is fixed in Circle 3.0 patchlevel 12 and above. Before patchlevel 12, you can fix it simply by manually creating the needed directories:

```
CD \Circle30bpl11
cd lib\plrobjs
mkdir A-E
mkdir F-J
mkdir K-O
mkdir P-T
mkdir U-Z
mkdir ZZZ
```

Object saving should then work. The syslogs are a different story; no data is written to the system logs because the code currently is configured simply to write all errors to the standard error file descriptor (stderr), and Windows doesn't seem to let you redirect stderr to a file the same way UNIX does. pl12 allows you to direct logs to a specific file instead.

4.9 When someone logs on to my Windows 95 MUD, the console screen gives: "gethostbyaddr: No such file or directory"

This means the MUD can't resolve the IP address of the connecting player's source site innto a hostname. You probably don't have DNS correctly configured in the Windows Network Control Panel menu (under configuration of the TCP protocol). Make sure you have the IP address of your ISP's DNS server listed.

4.10 My Mud crashed and my connection got closed. What can I do?

Just because your connection got closed from the mud (for example, if you get too much information sent to you and the telnet session gets closed), this doesn't always mean that the game itself crashed. Before reporting something as a crash bug, make sure that the game itself crashed, and above all, try to duplicate the circumstances before reporting it as a crash bug. You can also try using gdb to find out why the mud is crashing if it gives you a core dump.

4.11 Ok, so how do I use "gdb"?

GDB has some online help, though it is not the best. It does at least give a summary of commands and what they're supposed to do. What follows is Sammy's short intro to gdb with some bughunting notes following it:

If you've got a core file, go to your top circle directory and type:

> `gdb bin/circle lib/core`

If you want to hunt bugs in real time (causing bugs to find the cause as opposed to checking a core to see why the mud crashed earlier) use:

> `gdb bin/circle`

If you're working with a core, gdb should show you where the crash occurred. If you get an actual line that failed, you've got it made. If not, the included message should help. If you're working in real time, now's the time to crash the mud so you can see what gdb catches.

When you've got the crash info, you can type "`where`" to see which function called the crash function, which function called that one, and so on all the way up to "`main()`".

I should explain about "`context`" You may type "`print ch`" which you would expect to show you the ch variable, but if you're in a function that doesn't get a ch passed to it (real_mobile, etc), you can't see ch because it's not in that context. To change contexts (the function levels you saw with where) type "`up`" to go up. You start at the bottom, but once you go up, and up, and up, you can always go back "`down`". You may be able to go up a couple functions to see a function with ch in it, if finding out who caused the crash is useful (it normally isn't).

The "`print`" command is probably the single most useful command, and lets you print any variable, and arithmetic expressions (makes a nice calculator if you know C math). Any of the following are valid and sometimes useful:

```
print ch (fast way to see if ch is a valid pointer, 0 if it's not)
print *ch (prints the contents of ch, rather than the pointer address)
print ch->player.name (same as GET_NAME(ch))
print world[ch->in_room].number (vnum of the room the char is in)
```

etc..

Note that you can't use macros (all those handy psuedo functions like GET_NAME and GET_MAX_HIT), so you'll have to look up the full structure path of variables you need.

Type "`list`" to see the source before and after the line you're currently looking at. There are other list options but I'm unfamiliar with them.

(From Sammy <Samedi@cris.com>)

For more information, you can try checking out the *GDB Debugger* `<http://www.dgii.com/people/robertl/skunk/gdb/gdb_toc.html>`

4.12 How can I hunt bugs more effectively?

There are only a couple of commands to use in gdb, though with some patience they can be very powerful. The only commands I've ever used are:

```
run                     well, duh.
print [variable]        also duh, though it does more than you might think
list                    shows you the source code in context
break [function]        set a breakpoint at a function
clear [function]        remove a breakpoint
step                    execute one line of code
```

```
cont                        continue running after a break or ctrl-c
```

I've run into nasty problems quite a few times. The cause is often a memory problem, usually with pointers, pointers to nonexistent memory. If you free a structure, or a string or something, the pointer isn't always set to NULL, so you may have code that checks for a NULL pointer that thinks the pointer is ok since it's not NULL. You should make sure you always set pointers to NULL after freeing them.

Ok, now for the hard part. If you know where the problem is, you should be able to duplicate it with a specific sequence of actions. That makes things much easier. What you'll have to do is pick a function to "break" at. The ideal place to break is immediately before the crash. For example, if the crash occurred when you tried to save a mob with medit, you might be able to "break mobs_to_file". Try that one first.

When you 'medit save', the mud will hang. GDB will either give you segfault info, or it will be stopped at the beginning of mobs_to_file. If it segfaulted, pick an earlier function, like copy_mobile, or even do_medit.

When you hit a breakpoint, print the variables that are passed to the function to make sure they look ok. Note that printing the contents of pointers is possible with a little playing around. For example, if you `print ch`, you get a hex number that shows you the memory location where ch is at. It's a little helpful, but try `print *ch` and you'll notice that it prints the contents of the ch structure, which is usually more useful. `print ch->player` will give you the name of the person who entered the command you're looking at, and some other info. If you get a `no ch in this context` it is because the ch variable wasn't passed to the function you're currently looking at.

Ok, so now you're ready to start stepping. When GDB hit your breakpoint, it showed you the first line of executable code in your function, which will sometimes be in your variable declarations if you initialized any variables (ex: int i = 0). As you're stepping through lines of code, you'll see one line at a time. Note that the line you see hasn't been run yet. It's actually the _next_ line to be executed. So if the line is `a = b + c;`, printing a will show you what a was before this line, not the sum of b and c.

If you have an idea of where the crash is occurring, you can keep stepping till you get to that part of the code (tip: pressing return will repeat the last GDB command, so you can type step once, then keep pressing return to step quickly). If you have no idea where the problem is, the quick and dirty way to find your crash is to keep pressing return rapidly (don't hold the eturn key or you'll probably miss it). When you get the seg fault, you can't step any more, so it should be obvious when that happens.

Now that you've found the exact line where you get the crash, you should start the mud over and step more slowly this time. What I've found that works really well to save time is to create a dummy function. This one will work just fine:

```
void dummy(void){}
```

Put that somewhere in the file you're working on. Then, right before the crash, put a call to dummy in the code (ex: `dummy();`). Then set your breakpoint at dummy, andwhen you hit the breakpoint, step once to get back to the crashing code.

Now you're in total control. You should be looking at the exact line that gave you the crash last time. Print *every* variable on this line. Chances are one of them will be a pointer to an unaccessable memory location. For example, printing `ch->player.name` may give you an error. If it does, work your way back and print `ch->player` to make sure that one's valid, and if it isn't, try printing ch.

Somewhere in there you're going to have an invalid pointer. Once you know which one it is, it's up to you to figure out why it's invalid. You may have to move `dummy()` up higher in the code and step slowly, checking your pointer all the way to see where it changes from valid to invalid. You may just need to NULL a free'd pointer, or you may have to add a check for a NULL pointer, or you may have screwed up a loop. I've done all that and more :)

Well, that's it in a nutshell. There's a lot more to GDB that I haven't even begun to learn, but if you get comfortable with print and stepping you can fix just about any bug. I spent hours on the above procedure trying to get my ascii object and mail saving working right, but it could have taken weeks without gdb. The only other suggestion I have is to check out the online gdb help. It's not very helpful for learning, but you can see what commands are available and play around with them to see if you can find any new tools.

(From Sammy <samedi@cris.com>)

4.13 I just added n levels to my mud (from the stock 34). How do I set my imps up to level n without a pfile wipe?

You can write a quick and nasty function that will advance your imp (and imp only) to the max level (LVL_IMPL). This can be done with a quick idnum check so that only the original player (the first imp, he with id 1) can use the command to get to maximum level.

```
ACMD(do_upme)
{
  if (GET_IDNUM(ch) != 1) {
    send_to_char("You think IMP positions are that easy to come by?  Go Figure...\r\n", ch);
  } else {
    GET_LEVEL(ch) = LVL_IMPL;
    send_to_char("Advanced.\r\n", ch);
  }
}
```

4.14 I decided to wipe my pfile away anyway. What steps should I take to do this?

In order:

1. Shutdown the mud with "shutdown die" so that it won't restart.
2. Remove the player file, `lib/etc/players`
3. Restart the mud and login to recreate your imp character.

You should probably also remove files in plrobjs, unless you want the recreated characters to come back with the same equipment they had when they were deleted.

4.15 I want to expand the ability to pk in my MUD, allowing ASSASSINS that'll be able to PK without getting flagged. How can I do this?

The simple way to do this is to find all the "pk_allowed" checks and replace them with a `can_murder(ch, vict)` function call. Prototype the function in `utils.h`. Then, in `utils.c`, create a `can_murder()` function something like this:

```
int can_murder(struct char_data *ch, struct char_data *victim) {
  if (pk_allowed == TRUE)
    return TRUE;
  if (IS_NPC(ch) || IS_NPC(victim))
    return TRUE;   /* you can always kill these */
  if (PLR_FLAGGED(ch, PLR_ASSASSIN))
    return TRUE;   /* they can kill anyone */
  /* Add further checks here */
}
```

4.16 Why does it say "Connection closed by foreign host." and not display the "Byebye!" message I'm trying to send before cutting someone off?

This usually happens if you are doing something like this:

```
  send_to_char("Bye bye.  Come back soon, ya hear?", ch);
  close_socket(ch->desc);
```

The close_socket immediately dispatches/closes the connection, while send_to_char puts the message on the output queue to be dispatched next game_loop cycle. Therefore, the socket is gone. On some systems (ie old linux), this can even cause a infinite loop attempting to write to a closed socket. The proper way of doing this and other "Byebye" messages is to set the CON state of the player to CLOSE, like this:

```
  send_to_char("Bye bye.  Come back soon, ya hear?", ch);
  STATE(ch->desc) = CON_CLOSED;
```

This will then cycle to the next game_loop, dispatch the output queues (therefore sending the byebye message) and then close the socket. Further note, in some bizarre cases, this only seems to send about 40 characters and no escape codes. Sending more than 40 characters or escape codes (like the clear screen sequence) will crash the process reporting a problem similar to writing to a closed socket.

4.17 I run my mud on a unix system and the pfile/rent file works great, but on my home system it's all screwed up. What gives?

Some operating systems write binary data least-signifigant-digit-first, while others write it most-significant-first (big endian vs. little endian). Moving player files, rent files, mail files, and board files from one of these systems to the other will result in corrupted files. The solutions to this problem include:

- Don't bother trying to move those files.
- Edit your code to always write in one format.
- Develop a binary to ascii conversion tool (and ascii to binary) for all binary files.
- Develop an ascii read/write system to allow portability.

4.18 How do I get the CircleMUD to autoload when the Linux server is restarted?

In `/etc/rc.d/rc.local` find where things like sendmail and (maybe) gpm are started. Add something like:

```
cd /home/mudlogin/circlebpl12/
su mudlogin -c ./autorun&
cd
```

Of course, change the "`mudlogin`" to whatever the name of the account is that normally has control of the mud, and change the path in the first `cd` to whereever the mud is run from.

For more info: `man su`

4.19 My server shuts down my MUD everytime I logoff. How do I keep the MUD running when I logoff?

Instead of typing "`autorun &`" to start the autorun script (which starts and keeps the mud running), type "`nohup autorun &`". Running the autorun via nohup will keep the script and the MUD running when you logoff of the server. For more information type "`man nohup`" at the prompt on your server.

5 Code Changes for CircleMUD 2.20

5.1 How do I fix the bug where people can junk more coins than available?

Apprently in Circle 2.2, you can drop any amount of coins, and then be rewarded with more coins than you had in the first place. Here is the fix from Jeremy Elson:

Around line 480 of act.obj1.c, you will find the code:

```
            if (!str_cmp("coins", arg) || !str_cmp("coin", arg))
              perform_drop_gold(ch, amount, mode, RDR);
            else {
              /* code to drop multiple items.  anyone want to write it? -je */
              send_to_char("Sorry, you can't do that (yet)...\n\r", ch);
-->           return;
            }
          } else {
          ....
```

It should be changed to:

```
            if (!str_cmp("coins", arg) || !str_cmp("coin", arg))
              perform_drop_gold(ch, amount, mode, RDR);
            else {
              /* code to drop multiple items.  anyone want to write it? -je */
```

```
              send_to_char("Sorry, you can't do that (yet)...\n\r", ch);
            }
-->         return;
          } else {
          ....
```

5.2 How do I fix the "vstat" bug that crashes the MUD?

To the fix for the vstat bug, from Jeremy Elson:

In the file act.wizard.c, in the function do_vstat, in the mobile section of the switch (around line 1150), you'll find the code:

```
        mob = read_mobile(r_num, REAL);
        do_stat_character(ch, mob);
        extract_char(mob);
```

Add the line char_to_room(mob, 0) before extract_char(), like this:

```
        mob = read_mobile(r_num, REAL);
        do_stat_character(ch, mob);
        char_to_room(mob, 0);
        extract_char(mob);
```

5.3 How do I fix the "wizlock" bug that lets lower immortals lock out higher immortals?

Simple, insert this code into 'do_wizlock()' in 'act.wizard.c':

```
        if (value < 0 || value > LEVEL_IMPL) {
          send_to_char("Invalid wizlock value.\n\r", ch);
          return;
        }

+        /*   Do not allow people to wizlock above their level.
+         *   This bug came with Circle 2.20 source -- VampLestat
+         */
+        if (value > GET_LEVEL(ch)) {
+          send_to_char("You may only wizlock below your level.\n\r", ch);
+          return;
+        }
        restrict = value;
```

5.4 How do I fix the "mudlog" bug that lets people see me log in, even if I'm wizinvis?

For all the mudlog calls for entering the game, quitting, and so forth, you must change

```
mudlog(MAX(LEVEL_IMMORT, ...
```

to

```
mudlog(MAX(GET_LEVEL(i), ...
```

where "i" is either "ch" or "d->character" depending on the call.

6 CircleMUD 3.0 Questions

6.1 Are there any bugs in patch level 14?

There are no bugs. Only features. Seriously though, if perchance you find a bug, please mail it (along with a possible fix) to *the bugs database* `<mailto:bugs@circlemud.org>` and the CircleMUD list. Once in the bug database, it will be routed to the correct person. Note that no confirmation is sent back automatically. You can view the status of the bug at *the bugs database* `<http://bugs.circlemud.org/>`.

6.2 How do I access Online Creation?

Online Creation is not yet part of the Circle beta release. When it does become part of the release, it'll be accessed through a command called olc. Update: OLC will probably be in release 3.10 now, instead of 3.0. In the mean time, check out *the CircleMUD FTP Server* `<ftp://ftp.circlemud.org/pub/CircleMUD/contrib/olc/>`

6.3 How does the new bitvector system work?

The new bitvector system in CircleMUD 3.0 is an ascii based one. The old numeric values can still be used, but the new system should make sorting flags out substantially easier. The system uses an "a" as the value for 1, "b" for 2, "c" for 4, "d" for 8, etc. Once "z" is reached, the next letter in sequence is "A". Detailed information about how to use bitvectors with CircleMUD can be found in the *CircleMUD Builder's Manual* `<http://www.circlemud.org/jelson/cdp/building/>`.

6.4 When will the production release of Circle 3.0 be?

I don't know. Don't ask again. Same for the next patch level. If you must have bleeding edge stuff, check out the *CVS Snapshot* `<ftp://ftp.circlemud.org/pub/CircleMUD/cvs/>` until we get an online anonymous CVS. And don't ask when the next CVS snapshot will be, we'll do it when we have some changes worth it.

6.5 If someone logs in and just sits at the password prompt, the MUD hangs (i.e., no one else can connect or do anything) until the person enters their password.

Your system's POSIX non-blocking I/O might be broken. Look in the source file sysdep.h at the comment above the line that says "#define POSIX_NONBLOCK_BROKEN" for a possible fix. Once this is done, recompile the mud and try again. If you use the POSIX_NONBLOCK_BROKEN constant and it fixes your problem, please send mail to *the bugs database* `<mailto:bugs@circlemud.org>` and let us know exactly what kind of system you are using and what you had to do to fix it.

6.6 Why does CircleMUD use BUF switches all through the code, what's happening here?

From Jeremy:

This code is the new output buffering system that I wrote for Circle in the early (non-released) beta versions of 3.0. The old Diku code for queueing output (which stayed with Circle until version 2.20) was memory- and time-inefficient in many cases (and, in my opinion, was inefficient for the normal behavior of most MUDs).

First, I should explain what output queueing is and why it is necessary. On each pass through the game_loop(), the MUD performs a number of steps: check to see if there are any new players connecting, kick out people with bad links, read input over the network for all players, then process the input for each player that has sent a complete line over the net. The processing step is usually where output is generated because it is where MUD commands are processed (e.g., "kill" might generate output of "Kill who?") When output is generated, it is not immediately sent out to the player, but instead queued for output in a buffer. After all players' commands are processed (and each command generates the appropriate output for various players), the next step of the game_loop() is to send all the queued output out over the network.

The new output system that Circle now uses allocates a small, fixed size buffer (1024 bytes) for each descriptor in which output can be queued. When output is generated (via such functions as send_to_char(), act(), etc.), it is written to the fixed size buffer until the buffer fills. When the buffer fills, we switch over to a larger (12K) buffer instead. A "buffer switch", therefore, is when the 1024-byte fixed buffer overflows.

When a large (12K) buffer is needed, it is taken from a pool of 12K buffers that already been created. It is used for the duration of that pass through the game_loop() and then returned to the pool immediately afterwards, when the output is sent to the descriptor. If a large buffer is needed but none are in the pool, one is created (thereby increasing the size of the pool); the "buf_largecount" variable records the current pool size.

If a player has *already* gone from their small to large buffer, and so much output is generated that it fills even the large buffer, the descriptor is changed to the overflow state, meaning that all future output for the duration of the current pass through the game loop is discarded. This is a buffer overflow, and the only state in which output is lost.

Now that I've described how the system works, I'll describe the rationale. The main purpose for the two-tiered buffer system is to save memory and reduce CPU usage. From a memory standpoint: Allocating a fixed 12K buffer for each socket is a simple scheme (and very easy to code), but on a large MUD, 100 12K buffers can add up to a lot of wasted memory. (1.2 megs of memory used for buffering on a 100-player MUD may not seem like very much, but keep in mind that one of Circle's big selling points several years ago, when memory was expensive, was that it had a very small memory footprint (3 or 4 megs total!) And from a CPU standpoint: the original Diku used a dynamically allocated buffer scheme to queue output, which unfortunately meant that for *each* player, on *each* pass through the game loop, dozens of tiny buffers (often one for every line of output, depending on the code to execute the command) were allocated with malloc(), *individually* written to the system using individual calls to write(), and then free()'d. My system saves hundreds or thousands of calls per second to malloc() and free(), and reduces the number of system calls *drastically* (to at most one per player per pass through the game loop).

The trick is to choose the size of the small and large buffers correctly in order to find the optimal behavior. I consider "optimal" to mean that 90% of the time, most players stay within the limits of their small buffer (for example, when wandering through town or mindlessly killing some monster while watching damage messages go by). Hopefully, a large buffer switch is only necessary when a player executes a special command that generates an unusually large amount of output, such as "who", "read board", or "where sword". This critically depends on the fact that not everyone will be executing such a special large-output command at the same instant.

For example, imagine you have 10 players on your MUD. They are all wandering around town, and every once in a while one of them types "who", or reads the board, meaning that they are seeing more than 1024 bytes of output at a time. On such a MUD, I would hope that there would only be a *single* 12K buffer allocated which gets passed around among all the 10 players as needed. Now, all players think they can queue up to 12K of output per command without getting truncated even though only one 12K buffer actually exists – they are all sharing it.

But - there's a problem with this. There are certain cases when *many* players have to see a lot of output at the same instant (i.e. on the *same* pass through the game_loop()), all of them will need a large buffer at the same time and the pool will get very big. For example, if an evil god types "force all who"; or if the MUD lags for several seconds, then suddenly gets unlagged causing many commands to be processed at the same moment; or if 20 people are all trying to kill the same MOB and are all seeing 20 damage messages (more than 1024 bytes) on the same pass through the game_loop().

Unfortunately, the current patchlevel of Circle has no way to destroy large buffers so such cases are pathological and cause wasted memory. Unfortunately since I don't run a MUD I can't actually tell how often this happens on a real MUD. (If there are any IMPs out there who run large MUDs (say, >= 30-50 players on regularly), and you've read this far, please send me the output of "show stats" after your MUD has been played for at least several hours.)

Hopefully this clears up the way buffers work.

6.7 How do I add a new class? How do I add more levels?

Adding a new class is fairly easy in 3.0, in fact, someone has taken the time to put together a fairly complete document on adding a class, in this case a Knight class. The class.doc is available on *the FTP site* `<ftp://ftp.circlemud.org/pub/CircleMUD/contrib/docs/3.x/class.doc>`

The same applies for levels, in the file *levels.doc* `<ftp://ftp.circlemud.org/pub/CircleMUD/contrib/docs/3.x/levels.doc>` which can be found in the same place.

6.8 Is there a complete "coding.doc"?

No. Look at the README.NOW in the docs directory. It isn't complete, nor does anyone have the entire thing. It is being worked upon.

RTFM