mIRC 2: Advanced

This course will go into more depth with mIRC, teach you all sorts of quirks and ways to customize it, as well as greater detail in scripting.
As before, all notes will be made in *italics*, inputs and output will be in Times New Roman. All nick outputs will reflect your own, not my name. Any given keywords enclosed in the greater/less than signs are to be filled with the appropriate name (ex: <nick> is JaggedFel, or <chan> is #galactic_empire)

Chapter 1: Options

Alright, since you've obviously gotten mIRC to run, it's time for you to learn what the rest of the options do. For the most part, these are "set and forget", and I don't blame you. There are maybe three options I turn on and off, depending if I don't want to be disturbed. If I don't mention it, it's because I think you're better off leaving it alone, or it's self explanatory. If you really want to know for sure, guess what the help file's for?

IRC:
-Prefix own messages - Default is on, it sticks your name in front of what you type.
-Show mode prefix - Default off, nicks would include their status (op, voice, etc) with their nick
-Iconify query window - A good one. If someone /msg's you, it stays in the background instead of jumping to the front and interupting your typing (like AIM)
-Rejoin on connect - A little misleading, it should be "on reconnect." If you're disconnected from the server, and mIRC tries to auto-connect again, you'll be back in the same channels as before, in case they're not in Perform
-Keep channel open - If you're kicked or disconnected, mIRC won't close the window
IRC-Messages:
-Strip... Color - I highly suggest this one, it will make reading things a lot easier when around color-addicts
DCC-Ignore:
-Method - If you're going to be swapping files, I'd actually set this to disable. If you leave the defaults, it won't accept unless you tell it to anyways, so it's really not needed
Display-Options:
-Show in mIRC Title bar - Okay, so this one's self-explanatory. The only realy reason you'd need this is if you run multiple mIRC windows (I run three).
-Windows... - Great little feature, allows you to minimize mIRC, but keep some windows open, like channels, PM's, or the script editor.
-Tray... - Again, great little thing to get mIRC out of the way when minimized, instead of sucking up Start bar space
Other:
-Confirm... - Adjust these as you see fit
-Keys... - Again, but the only thing you'd probably want to add is the Escape key.

Chapter 2: Commands

Okay, I'm going to add more commands that you will use for scripting and such, with examples in the next chapter. Commands unclosed in < > are required, while [ ] are optional. And as stated in mIRC 1, the help file is your friend.

/enable <group> - This is used to turn on a group of remotes enclosed by **#<group> <on/off>** and **#<group> end**. Helpful for scripts you don't need 100% of the time.
/disable <group> - Turns the group off
/dns <nick> - Mainly for those of us that play online and setup games in mIRC, this will uncover the IP under the person's hostmask. *Note: does not work for AOL users, and may not work for users with routers*
/timer [-m] <repeats> <duration> <command> - Great little tool here. The optional -m switch

changes duration to milliseconds, instead of seconds. Repeats is the number of times the timer will fire. If set to zero, it will loop indefinately until turned off. The Command is a single command. The use of brackets does not work. If you have multiple commands, it's best to put those into an alias and call that. *There are more switches, feel free to look them up if you need them.* A simple one would be /timer 1 60 /msg $chan SPAM! That will just send the message "SPAM!" to the channel once after 60 seconds.
/echo [color] [-ag] [channel] <message> - Echos are just that; replies from yourself that do not get sent to the server, normally used to notify that a script has fired successfully. The [color] option allows you to obviously change the color of the echo. The default can be changed in the colors dialogue. The -a switch causes it to echo in the current window, instead of the server window, where it will probably go unnoticed. -g will prevent the echo from being logged if logging is enabled.
/set <%var> <value> - This is one that you probably use a lot in complicated scripts. Variables are always preceded by the % character, and cnbe set to whatever you want. Unlike other programming languages, you can change a variable from text to numbers with no difficulty.
/inc <%var> [value] - Using no defined value will just add 1 to the variable, while changing the [value] will add that to the variable.
/dec <%var> [value] - If inc increases, then dec must...
/goto <label> - *Scripting command* - Best demonstrated, this allows you to skip sections of code, or go backwards to create loops

And not really a command, but a function, the If function:
if (<value1> <op> <value2>) { <commands> } - You will learn this one very quickly, as you will probably use it in most of your scripts. <op> is anything like "equals" (==), "less than" (<), "less than or equal to" (<=), to some new ones like <nick> **ison** <channel>, or <text> **isin** <string>. For the full list of operators, it's best to use the help file for this one, there's a lot of info.
There are also some functions, denoted with a **$**. Ones that you will probably use are **$right (<string>, <value>)** and **$left** (same as $right). $right reads <value> characters in <string> from the right, $left from the left. **$calc(<equation>)** is used for mathematical stuff, such as (1 + 1) or (10 / 2), and you can place variables in there instead. **$len(<string>)** will return the number of characters in a string.

Chapter 3: Pop-ups and Users

I'm going to go into detail about two of the other tabs in the scripting editor, namely Pop-Ups and Users (you can't do anything in Variables, it just stores stuff there).
Pop-ups is where you can change the menus that you get when you right-click, and you can change Commands on the menu bar.

In the Pop-ups tab, if you click View on the menu bar, you'll see a list of menus you can change. In all of these the format is **<title> : <commands>**. As you can see if you open up the tab is if you want to stack commands in a sub-menu, start the line with a period. Seperators are a single dash on a line.
The menus you can edit are Status (server window), Channel, Query, Nick list (or nick, same thing), and Menu Bar (the Commands menu in the main window).
This is where you can use **$1**, such as when you right-click a nick, in which the nick will be stored in $1 by mIRC, or **$?="<question>"**, in which an inputbox will come up asking for whatever you want to add. Look at the existing Pop-ups to get a better idea. *Note: Don't forget, you can make those $$ so the script will not finish unless a value is assigned.* That's pretty much it, what you do here is up to you.

The Users tab is mostly a storage tab, where nicks, usernames, hostmasks, and/or IPs are stored and assigned a user level, allowing better control of remotes. You can either add them by hand in the format **<level>:<user>**. <user> can be in the format of just <nick>, or <nick!username@hostmask>, with wildcards anywheres in there. Ex: 5:*!Bob@* This will make the person with the username Bob level 5, no matter what his nick or IP/hostmask is. There are

really three commands that you can use to play with users; **/auser <levels> <nick/address>** which uses whatever you enter, **/guser <levels> <nick> [type]** in which you can just enter the nick and the hostmask type (for the full listing, again use the helpfile), and **/ruser <levels> <nick/address> [type]** which removes the indicated levels for that nick/address/mask. In all of these, you can have multiple levels, seperated by commas. When adding, realize that a level 5 has access to 1, 2, etc. If you want them to be 5 only, and nothing else, add **=** before the user level. You can use userlevels to control who fires what remotes.

Chapter 4: Some scripts

Alright, so I've given you a bunch of new commands, and a whole 'nother tab to play with. To start off, I'm going to give you a couple scripts I found that I really like. The first one will echo all of the colors in mIRC, and the second will create a new window filled with ASCII characters and the keycodes to use them. Simply type /colors and /ascii to use them, respectively, after you've copied them into your Aliases tab.

```
/colors echo -a Colorbar: $chr(3) $+ 0,0 $+ $chr(3) $+ 1,0 0 $chr(3) $+ 0,1 1 $chr(3) $+ 0,2 2 $chr(3) $+ 0,3 3 $chr(3) $+ 0,4 4 $chr(3) $+ 0,5 5 $chr(3) $+ 0,6 6 $chr(3) $+ 0,7 7 $chr(3) $+ 1,8 8 $chr(3) $+ 1,9 9 $chr(3) $+ 1,10 10 $chr(3) $+ 1,11 11 $chr(3) $+ 0,12 12 $chr(3) $+ 0,13 13 $chr(3) $+ 0,14 14 $chr(3) $+ 1,15 15 $chr(15)
```

Yes, that is all one line.

```
/ascii {
  window -l -t15,30 @ASCII -1 -1 400 300
  aline @ascii Character $+ $chr(9) $+ ASCII value $+ $chr(9) $+ Alt+
  VAR %a = 32
  WHILE (%a <= 255) {
    aline @ascii $chr(%a) $+ $chr(9) $+ %a $+ $chr(9) $+ $base(%a,10,10,4)
    INC %a
  }
}
```

Okay, the colors script will echo Colorbar: followed by the numbers 0 through 15 highlighted by their respective colors. The $chr(3) is what's inserted when you press Ctrl+K to get colors normally. The numbers in this case are in the format **foreground,background text**. Colors are cool to use once in a while, but overuse can lead to action against you in a channel, as overuse is probably against the channel's rules.

The ascii script is obviously a little complicated, and to tell you the truth it took me a while to figure it all out myself so I could use it elsewhere. *Note: you'll notice that the commands in the ascii alias are not preceded by "/". This is already implied in the scripting editor, and are not required. It's still good practice to use it though.* There are actually 40 different switches, broken up into three groups. The ones used here are Listbox, and Tabs (spaced as 15 and 30). **@ASCII** is the window name, the **-1**'s mean default x and y location for the window, and the last two numbers are height and width. The **/aline <window> <text>** command adds a line of <text> in the specified window. After that comes the While loop. **VAR** is the variable initializer, in this case the first %a is 32. The main part of the while loop comes after that, in the script will continue to loop between the **{ }** as long as %a is less than or equal to 255. When it reaches 255, it will run one last time, then continue with the rest of the script (which in this case is nothing). So there's a few more functions for you to play with.

Chapter 5: Scripting

Alright, by now you have the basic tools to do pretty much anything you want. This section is going to focus on users, if statements, and some funtionality scripts.

Okay, so you have a bunch of people that you've set at different user levels, now what? With the remotes you've seen so far, they've **on 1**. That "1" is the default user level, meaning that the remote will fire for everyone. A user at 100 can access remotes that are set below 100. So if there's a remote that's **on 200**, it will ignore the 100 user. This lets you customize scripts to respond differently to different people. Here's a quick example. *Note: again, it's best to not actually use this one ;-)*

```
on 1:JOIN:#:{ /notice $nick Welcome to $chan $+ ! }
on 5:JOIN:#:=
on +100:JOIN:#:{ /notice $nick Welcome to $chan $+, oh Important One }
```

First line is the base. Whenever anyone joins, /notice them. Simple. What the second line says is "for anyone level 5 or higher, don't do anything." That's what the equals sign does. And the third line says "if the user is exactly level 100, notice them." The plus sign lets you single out a user level, and nobody else will be affected. So all in all, the only userlevels that will see anything are 1-4, and 100.

```
on @1:text:*poop*:{ /kick $nick Potty mouth }
```

In this case, I've used the **@** control, which means the remote will do nothing unless you are opped, which in this case will kick the offender. This is used mainly for scripts that involve opping, kick/banning, or anyhting else you physically can't do without ops. Also prevents those "You are not a channel operator" messages if you don't put the @ prefix on.

```
#1337 on
on @+2:text:*:#:{ set %kick no
  if (n00b isin $1-) { set %kick yes }
  if (1447 isin $1-) { set %kick yes }
  if (%kick == yes) { /kick $chan $nick No 1337 5p34k! }
}
#1337 end
```

This is a shortened script from my bot. Starting from the beginning, we have the group opening. This lets mIRC know that anything following **#<name> [on/off]** is part of that group, until we reach **#<name> end**. Second line uses both the @ and + prefixes, so if you're an op, and a strictly level 2 person speaks, start off by setting the variable. The next two lines check to see if certain words were in the spoken sentence, in if they are, change the %kick variable. The last line checks the variable, and kicks the offender if it returns "yes". This script gets used a lot, and is actually kinda funny to watch.

Last one for today: *This one may get you in trouble, depending on where you use it.*

```
/mute {
  set %i $nick($chan,0)
  :next
  if ($level($address($nick($chan,%i), 2)) < 5 && $nick($chan,%i) != $me) { /notice $nick($chan,%i) $1- }
  dec %i
  if (%i > 0 ) { goto next }
}
```

First of all, the purpose of this script is to be able to talk to people in a channel even though it's +m or moderated. Don't actually use this unless you know they won't care.

Okay, the first line is just the alias name, yay. Second line is new to you. **$nick($chan,0)** gets the total number of nicks in a channel. We store this in %1, to be used as our counter. **:next** is out label name for the goto later on. Now programmers hate "goto", it can create messy code. This is small, not important, and chances are you won't even use it. If you're doing something big, try to use loops, but sometimes you just can't. Fourth line is where all the important stuff happens. Working from inside the **if ( )**, We get the user level of the address belonging to the % i'th person in $chan. Got it? Simply put, find out what user level that person has. If it's less than 5, and it's not yourself (that's what the **&&** does, it means "and" inside the ( ). You can also use **||** for "or"), then /notice that person whatever you typed after /mute, so like /mute Haha, I'm still

talking! *Again, don't actually do that...*
After that, you decrease %i, *yes you're working from the bottom up*, and if %i is still positive, go back to :next and start over again.

Well, you basically have everything you need now to do almost whatever you want. Again, I can't stress enough that you read the helpfile before going around asking other people. Believe me, if you figure that out now, you'll be much better off later on. Until next time, happy scripting!