



Mirc script comment

! Cookie - gives someone a cookie. ! Pie- Gives someone a pie... Calculated - [Number[\*]-+|/]Number] ===== Scenario based on Diamond Server: IRC. SwiftIRC.net Channel: #Diamond == ALIASES REQUIRED: ===:) \$upper=FUNNY COMMANDS: === Cookie/Si:#: { if (\$mainbot ison \$chan) & amp;& amp; {\$me != \$mainbot} { halt } else { set %cookie.who  $iif(\$ieft(\$2,1) == \$null, \$ick, \$2-) describe \cookies provide \cookies$ \$mainbot) { halt } elseif (!\$2) { set %ctarg1 \$iif(\$left(\$1,1) == @, msg \$chan, message \$nick) %ctarg1 \$logo(error) Provide me several figures to calculate. unset %ctarg2 } } Chapter 2: Getting started You have reason to use mIRC and script editor. Now we can start learning how to script. Read the following section and follow the examples. You will need to understand the skills in this section to move and make any sort of script whatever. This is a very long section, especially for beginners, so it's time to read this. Script types[edit] There are three types of scripts in mIRC.; nicknames, pop-up and remote controls. Each of them consists of different parts and has its own tab in the script editor. Aliases are typically used to simplify everyday IRC tasks, such as identifying a nickname. The pseudonym syntax is as follows: An example of an alias placed on the script editor's alias tab would look similar: hello say Hello, I am \$me and am a mIRC user. The first part of the alias is hello. This is the alias name. At the command prompt, type the /hello command. Say hello, I am \$me and I am an mIRC user. part is a command that runs when you type /hello. After typing / hello and pressing Enter, the line would appear in the current window, saying hello, I am mynickname and I am an mIRC user. One important thing to look out for is that I did not use / before the nickname name or command. This is not necessary in the script editor. mIRC prefixes each script command with a //that each command and automatically evaluates all team variables or identifiers. So, mIRC would treat both of these exactly as well: / hello / say hello, I \$me and I am an mIRC user. Hello, I \$me and I am an mIRC user. Hello, I say hi, I \$me and am an mIRC user. Hello, I and } be able to add everything that includes multiple rows. For example, the nickname hello that we had above could use/say twice. This can be done using the following code: Hello { say hello, I \$me and I am a mIRC user. WARNING: Another way to perform multi-command blocks of code mIRC script called piping is when the pipe symbol () is stuck between two commands such as /echo hi! | /echo goodbye! It's hard to read code, takes mIRC longer to process, and generally has bad coding practices. Use it sparingly. Function key aliases [edit] Function keys can also be used by aliases. Under Aliases, type this script in the script editor. F1 {var %n = \$?=What is your nickname and password? You should see a little dialog box popup asking for your nickname and password. Enter nickserv and password if you have one, and press Enter. Please note that depending on your network, you may not have a pseudonym service. If 12 Fkeys are not enough for you, well, you can assign up to 36 teams to Fkeys. Use c or s behind the key to make the alias work for shift+FX or Ctrl+FX. sF1 { var %n = \$?=What is your nickname and password? That's the only thing that's changed since the last example. However, instead of pressing F1 to activate it, hold down the shift and press F1. The same should happen as last time. Another notable thing about functional key aliases is that you can use them at a command prompt by entering an alias name, such as any old alias name. For the nickname we made, just type /sF1. Aliases in remote scripts [edit] aliases also serve a much more important purpose than just a set of commands. Remote script aliases can be used to truncate code and reuse a certain set of commands, making writing code much easier. Aliases change their syntax slightly when they are in remote scenarios. Alias aliasname command(s) To perform an alias in a remote section, it must have an alias prefix. For example, a team that is just made above, now it would look like a nickname hello { say hello, I am \$me and I am an mIRC user. The alias prefix is the only thing that changes. Alias launching still works the same way; type /hello to see text strings on the screen. The real advantage of putting a nickname inside a remote script is the possibility reuse it. For example, you might have an alias that increases the number three times. Technically, you can use the /inc command to do this, but for the example we will make our own nickname. alias incnum3 { // \$1 is the first paramint transmitted by /incnum3. BAR So if you are using /incnum3 7, \$1 = 7 var %x = \$1 incnum %x incnum %x incnum %x echo %x } alias incnum { inc %x } See how I used the nickname three times? If this was a longer script, I could have saved a lot of time and write using aliases several times. People familiar with other programming/script languages will notice that aliases in this regard are similar to features or subroutines that can be recycled. Local aliases[edit] If you want a command to run in only one script, or it can overwrite a command, built-in mIRC or other script, you can perform the alias -I before the alias name. -I is a switch that specifies an alias incnum3 { // \$1 is the first parameter passed to /incnum3. BAR. So if you're using /incnum37, \$1 =7 var %x = \$1 incnum %x incnum %x echo %x } Alias -I incnum %x echo %x } You cannot now use /incnum3 can call it. Recursive aliases [edit] mIRC cannot call an alias from the inside. For example, this code won't work: alias echomyname { echo -a Hi I'm \$me \$+ ! echomyname } You can't use /echomyname ? { echo -a Hi I'm \$me \$+ ! lt's a recursive alias. echomyname } It would line forever, and most likely crash mIRC. Fall programs are usually a bad idea. And it's quite embarrassing for you when your program locks an infinite loop. So don't do that. Now there is one warning for this recursive pseudonym business. Tell me that you wanted to write through an alias with your custom. When you call an alias from the inside itself, and it has a built-in version of mIRC, the alias will call the mIRC version. Take a look at this example: alias k { inc %k kick \$1- Total kicks %k } Let's say you wanted to rewrite the kicking nickname to tell you how many strokes you've made. You can do /k from now on. Or, using the message that mIRC can't duplicate aliases, you can do this: alias kick { inc %k kick \$1- Total kicks %k } This will increase your future number and future user you want from the channel, with total kicks message. This is very useful for more advanced scenarios, especially in creating themes for full scripts. [edit] Pop-up menus are user-defined context menus that are displayed in certain mIRC locations when you right-click. There are five pop-up menus that you can customize; one menu menu one in the status window, one in the nickname, one in the channels, and one in private messages. [edit] Open the script editor on the Pop-up window tab. For the purposes of this book, I will use the Channel tab. To define a simple menu item, the syntax is as follows: So to create a simple menu item that would change your nickname to mIRCUser, you can do this: Replace Nick: Nick mIRCUser [edit] Now right-click any channel. You should see the option saying: Change to mIRCUser. But what if you have more than one nickname that you want to use. Maybe you have aliases mIRCUser, Cakeman and Blackknight. You can add a submenu to this existing menu. Change Nick to .mIRCUser: nick mIRCUser. Cakeman. Blackknight: Nick Blackknight all three of your aliases. You can even make a submenu inside your menu. Change Nick to . Nickname.. mIRCUser: nick mIRCUser .. Cakeman.: Nick Blackknight [edit] But what if your server has a nickname service and you identify with your aliases as an example of a fkey alias? Well, pop-up menu definitions can contain multiline sections, as in other parts of the script using { and }. Change Nick to . Nickname.. mIRCUser: { nick mIRCUser ns identify password } .. Cakeman ns identify password } .. Cakeman ns identify password } .. Cakeman ns identify password } ... affected. This is specified in the identifier \$1. Note that these menus go below the Nicklist view on the Pop-up menu. Kick \$\$1: kick \$\$chan \$1 You have kicked \$me \$+! Insurance \$\$1: mode \$\$chan \$1 You have kickban \$1: ban -k \$\$chan \$1 2 You have kickban \$1 2 You have kickban \$1 2 You have kickban \$1 in this case means the same as \$1. This is the nickname selected in the nickname (if there is one and you are in the channel). When you click on the option (kick, insurance, or kickban), it will make this option, no matter what nickname is \$1. For example, the future option starts at \$1 from \$\$chan with the message: You have to kick mynick! Now let me use this example to explain the double \$\$ prefix. When used with an identifier, this will stop the command from running if the identifier \$null. For example, if you don't select a nickname for a nickname be hidden from view if you did not select a nickname in a nickname. Nicklist menu also supports everything menu support. You can use multi-line commands, submenus, and more. They work exactly the same way. The only real caveat using the pop-up menu that is directed to the user on the nicklist is to remember to use \$1 or \$1. [edit] The menu menu on the menu works just like other pop-up menus, unless they appear in the menu bar mIRC windows at the top of the Commands menu. If you want, you can also change the none of the Command menu. In the first row of the pop-up file, place the word Pie below the menu bar section. Save the file and look at the menu bar. Now the menu should be marked as Pie Chart. Menu item identifiers and variables[edit] Menu item variables and identifiers are evaluated each time you open the menu. As can be seen from the previous example, a nicklist menu like this: Kick \$1: kick \$\$chan \$1 You have kickbanned by \$me \$+ ! Do a \$1 estimate every time you open the menu item will be displayed, but grayed out and disabled. If you only have a set of %mytext, the item will evaluate \$null and the menu will not appear at all. The advantage of using identifiers and variables in pop-up menus is to make them more dynamic, as shown in the example of the kick/ban menu above. [edit] Similar to aliases, pop-up files can be defined in remote files in almost the location of the pop-up window; status, query, menu bar, or nickname. Unlike the Pop-up Menu tab, the Menu under the Remote tab does not know where to open the menu, so you have to say this after the menu prefix. Menu Channel { Get Channel { Get Channel { Get Channel } It's about the same as the default menu on the MIRC pop-up/Channel tab, but it is now in the remote file that has the prefix of the menu attached. menu channel, query { Slap Something!: I get slaps a random user with a large trout. } The remote file menu may also have multiple menu is used to change the appearance of menu items. You can use menu styles to select or deactive a menu item. There are three menu styles that you can use using \$style(2) Disabled item off element \$style(3) Checked and disabled item  $\sqrt{Checked}$  and disabled This is an example of a menu that uses styles to disable items if you are not connected to the server, style(2), style(1)) Connect #mIRC siif(!\$server, style(2), s check marks ( $\sqrt{}$ ) next to them. [edit] \$submenu is a pretty advanced identifier that makes a dynamic submenu pop-up menu. It calls a custom identifier that increases every time it passes through an identifier it passes through an identifier with the parameter \$1, where 1 is a number that increases every time it passes through an identifier; the channel list. alias infochan { if (\$1 == start) return - if (\$chan (\$1) ischan) return \$ifmatch: .msg chanserv info \$ifmatch if (\$1 == end) return - }; Verify that ChanServ \$network if (\$notify(ChanServ) & gt; 0) { var %csison = 1 } elseif (\$notify(ChanServ) & gt; 0) { var %csison = 0 }. timer 1 1 .notify -r ChanServ return %csison } art status { ; If Chanserv is not available, hide the menu. \$iif (\$ cschk < 1.\$style(2)) Receive channel information ; Otherwise, list the channels. .\$submenu (\$ infochan(\$1)) } The most important thing to note is how \$submenu is used. \$submenu is used. \$submenu is used. \$submenu is used. \$submenu(\$ infochan(\$1)) } The most important thing to note is how \$submenu is used. \$submenu(\$ infochan(\$1)) } it expires. So \$chan (\$1) in the custom identifier \$ infochan will return the first channel you've logged on, then the second, and so on, until the channels are over. Let's say you're #mIRC #Windows, and #Scripting, and ChanServ is on your network. If you right-click your status window, you will see the Get Channel Information menu  $\rightarrow$  and three menu items in the submenu of that menu; #mIRC, #Windows and #Scripting. Clicking on one of them will make /msg chanserv information channel. This command provides information channel if it is registered channel. This command provides information channel. This command provides information channel if it is registered. ChanServ- Description: Help Channel mIRC Scripting - - ChanServ- Registered: November 14 20:24:34 2005 CST - - ChanServ- Last used: Mar 2 4 11:0 2:55 2008 CDT - - ChanServ- Theme is set by: Master Scripter - - ChanServ- URL: - - ChanServ- E-mail address: someemail@email.com - [edit] To separate menu items, between the elements, you can use one - (hyphen). Remote scenarios [edit] Remote scenarios account for most of what you script mIRC. They are a combination of aliases, pop-up menus, and event-driven command blocks. Unlike aliases and pop-ups, you can load multiple remote controls at once. Each remote file usually has one script. A separate script is a remote file (a file that ends with .mrc or .ini) with related events, aliases, and pop-up menu definitions. Scenario example be a script that allows you to connect to a server or manage a list of citations that people have told the IRC. Basic syntax the event looks like this: Remote events[edit] On \*: Event:parameters: { commands } \* \* are the kind of user-level event to be activated. User levels in this section are defined later. Typically, the event will cause anyone with a user level of 1 or higher (1 is the default level of mIRC unless you change it, so it also usually means something). The event is the name of the event that you define. Examples of event so that they can work correctly. Part of event settings can be anything. This usually corresponds to the text where the event takes place, both or none of them. This event block contains all the commands you want the script to do. About \*: Text:\*:#:{ if (hello isin \$1-) { msg \$chan Why hello, \$nick \$+! } } This is an example of one of the most popular remote events and one of the most popular remote events and one of the most easily understood. It triggers when someone says a text that matches matchtext. Matchtext in this case is \*, a wild card symbol. That means he'll match anything. If the then-next statement of the body of the event determines what text it matches. This is a better idea in practice using one event that uses if-then-else sentences will ensure that everything is processed in the right order. In this scenario, the script checks whether the word hello is in the text, but only in the channels. # after \* is what determines where the event is caused. You can also specify ? only for private messages and \* to activate it on both channels and PM. \*:Load: { echo - a Thank you for uploading my script! } This is another example of a remote event. This one does not have parameters. The event simply triggers when you load the script in which it is located. When you upload this script, a line will appear on the screen thank you for uploading my script!. One important thing to know about remote event scenarios is that most of them don't work when you activate them yourself. Some events, such as On Kick or On Ban, and others, such as On Text, won't work when you say something on your channel by uploading a script. mIRC usually tell the help file when you can or can't trigger it yourself. Otherwise, it's a test and a mistake. CTCP events [edit] CTCP events for a nother client. A CTCP events can be used to override the default mIRC action (except VERSION) for the CTCP event, but can also be used to provide your own CTCP responses. CTCP \*:secretword! } still { msg \$nick Sorry, you're wrong. } \*:ctcpname:\*/#/?:{ command } CTCP event looks very similar to remote event. However, instead of the prefix, CTCP events begin with CTCP. \* is still the same; access level required to trigger the event. Ctcpname is the name of the ctcp that the event should trigger. This can be anything from built-in CTCP events, such as TIME or VERSION, or a evasive event, such as TIME or VERSION, or a evasive ev ? is PM, and \* is both. CTCP \*:secretword:\*:{ if (\$2 == moo) { msg \$nick You have a secret word! } still { msg \$nick Sorry, you're wrong. } This event will answer any time someone CTCPs you text secretword. If it matches, the script will be pm to tell them. CTCP \*:%secretword:\*:{ msg \$nick You have a secret word! } You can change the secret word and truncate the code at any time by using the variable (%secretword) as the equivalent of a CTCP event. Now the script will respond only when they ctcp you get the correct word. CTCP \*:Version:\*:{ tccpreply \$nick VERSION MyScript version 1.0 } Although you cannot hide the initial mIRC version response, you can add your response to the standard one. This one will send a CTCP response back using /ctcpreply to the person who CTCP'd you with your custom version to reply, followed by a mIRC one. CTCP \*:\*help\*:\*:{ msg \$nick What can I do to help you \$nick \$+?} By covering match text with wildcard characters, this script will trigger any time someone expresses a word to place anywhere in the CTCP request and msg them back. Raw events [edit] Raw events are extended events that trigger when you receive a back response. Raw numeric:matchtext:{ commands } Unlike remote controls and CTCPs, raw events are activated only in your client when you receive raw numbers from an IRC server; for example, using /whois or /away. Raw 431:\*:{ .haltdef echo-s ERROR: Raw 431 - Invalid nickname. } This event replaces the mIRC default response with Raw 431, the message that you receive when you do not specify a destination for a command, such as /whois or /nick. It silences the default response by using /haltdef. You may know how to use remote events now, but they are only a small part of creating remote scripts. Making events yourself is easy, but there are many teams put inside events. This is the actual challenge of the MIRC scenario, and what the rest of this book will cover. Variables are assigned a certain value and they always return the current value when they are extracted. variables are not recorded in the scenario. This means that the variables of variables are two types of data; text, numbers or any of these two. Variables are created during scripts and do not record other local variables or global variables of the same name. Local variables are destroyed after the call script is running. alias saysomething { var %text = \$\$?=What should I say? echo -a %text } If you enter /saysomething and press OK. This text will be stacked in %text, and then everything that is %text will be repeated on the screen. %text will not be available at the end of the script. Try typing //echo -a% text at a command prompt. You should get \*/echo: not enough parameters error. This is because variables [edit] Global variables [edit] Gl % connnick \$?=What should be your nickname? } nick % connnick is. Now open the script editor and open the Variables tab. You should see a variable called % connnick is. Now open the script editor and open the variables tab. You should see a variable called % connnick is. Now open the script editor and open the variables tab. You should see a variable called % connnick is. whose value is any nickname you have selected. You can also see other variables if you're running other scenarios. All of these variables are global and will remain under the Variables are global and will remain under the value until you use the /unset command. \*: Connect: { if (!%connnick) { set -u0 %connnick } Notice the difference in this scenario? The variable %connnick will, in some way, act as a global variable in this scenario for the -u0 flag in the /set command. -uN make global variable scattered after N seconds. However, if you specify 0 N, the variable will be disabled when the script finishes. It is not quite like a local variable; other global variables may overwrite it and vice versa. However, this is still a way to make sure that the script clears all its global variables when it ends. Mathematical equations with variables; commands /inc and /dec and = (assignment operator). alias incnum2 { var %num = \$\$1 inc %num inc %num echo - a Your number has been increased by two! You have 9! alias decnum2 7 you will get a message your number has been increased by two! You have 9! alias decnum2 { var %num = \$1 dec %num dec %num dec %num ber has been reduced by two! %num \$+ ! } The /inc commands have now been changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. The same /incnum2 7 changed to /dec commands. This will reduce your Two. code snippet uses the assignment operator to perform math with a variable. The first thing this code does is check if you gave it three parameters. It ensures that \$1 is the number, \$2 is a mathematical operator (+, -, \*, or /), and \$3 is the number. If they are all correct, the script adds two numbers to variables %num1 and %num2 have undergone an operation (regardless of what they put in USD 2) and the result is reported. Doing command / math 3 \* 2 would output 6. alias square { var %1 = \$\$1 var %2 = %1 \* %1 echo -a %2 } A simpler alias than the last, this command simply takes the first number you give it and multiplies it by yourself. /square 3 will give you 9. Identifiers[edit] Identifiers are similar to variables. They are dynamic in scenarios that measure when they are analyzed in a scenario. However, unlike variables, they can measure \$null. For example, one identifier that we've used so far is \$1. \$1 evaluates the first parameter, if any, transferred to the team when it runs. However, this is not always the same. In the example above, we could use the /square 3 command. \$1 in this case is 3. Let's change this from 3 to 4. Now \$1 is 4. Pass anything on it, and \$1 will be valued \$null. Another popular identifier used \$chan. \$chan any channel script or command runs. Make sure that two channels are open. Go to

one of them and type //echo \$chan \$chan. The effect of this action is printed in the current channel (the first event of the \$chan) to the right name of the second \$chan). Now go to another channel and follow the same command. Note that you still receive the channel name. This is because\$chan re-evaluates every time a script occurs. Built-in identifiers[editable] Built-in identifiers are identifiers encoded to mIRC. There are dozens of built-in identifiers, but some of the most used are \$nick, \$chan, \$me, \$1 and \$+. Identifiers built-in mIRC have priority over custom ones and cannot be changed or recoded, unlike built-in commands. Custom identifiers[edit] A custom identifier is an alias that can return a value. Custom identifiers are a common alias, but are used differently. They are called the same \$ prefix as the usual identifiers, not the /prefix used for calling as a team. alias addnum { if (\$\$1 isnum) & amp;& amp; (\$2 isnum) { return \$calc(\$1 + \$2) } } The script restarts for the first time to verify that both parameters are the correct numbers. Then it returns the value of the two numbers added together. Value. the way to use this is //echo -a \$addnum(n1,n2). If you text as a team, nothing will be displayed because it returns the value instead of replicating it. However, you can change this by using an identifier \$isid. alias addnum { if (\$\$1 isnum) & amp; & amp;; (\$2 isnum) { \$iif(\$isid, return, echo) \$calc(\$1 + \$2) } \$isid returns \$true if the alias was called as \$addnum has been called as \$addnum and \$false if it was called as an alias /addnum. Together with another identifier, \$iif, we can determine \$isid \$true \$false or not, and take action based on what the result is. If \$isid true, it means that the addnum has been called as an identifier and returns the value. If \$isid is incorrect, then the addnum has been called an alias and will replicate the value on the screen. Try the team in both directions now with 1 and 2. Both times you should get 3 custom properties[edit] You may notice that some identifiers have properties. For example, if you insert //echo -a \$chan(1).topic, the theme of the first channel you're on the screen will appear. .topic is an identifier property. You can also add properties to identifiers. alias addnum { if (\$\$1 isnum) & amp; & amp;; (\$2 isnum) { \$iif(\$isid,return,echo) \$iif(\$prop == neg,\$calc(-1 \* (\$1 + \$2)),\$calc(\$1 + \$2)) } Again, using \$iif, we can check the identifier property. If you use the //echo -a \$addnum(1.2).neg command, you will receive -3 instead of return 3. This is because the .neg property was used with \$addnum. Note: MIRC built-in identifiers with the same name. You cannot restart the script \$chan return 42. mIRC will still return the name of the channel you are using. [edit] Comments are a very important part of coding when your scripts start to get very large and complex. Hopefully tomorrow you will remember what your code did last night, but can you say you'll remember what the code lines that are ignored by an mIRC translator and can be used to remind you what a particular part of the script is doing. Here's what the last example might look like if we commented on it. [edit] nickname addnum {; Are the \$1 and \$2 numbers? if (\$\$1 isnum) & amp; (\$\$2 isnum) {; Has the addnum been called as a command or as an identifier? ; Should the number be negative? \$iif(\$isid,return,echo) \$iif(\$prop == neg,\$calc(-1 \* (\$1 + \$2)),\$calc(\$1 + \$2)) }; End of code } All comments in this code were comments from one line. One-line comments begin or ; or // and a blank space, then a comment. We were also easy to use // comments in this scenario instead of ;. All comments in one line end at the end of the line from which they begin. [edit] Multiline comments include Lines. They start wherever the code is /\* and ends where it is in the \*/ code. alias addnum { /\* Are \$1 and \$2 numbers? Addnum has been called as a team or Should the number be negative? \*/ if (\$1 isnum) & amp; (\$2 isnum) { \$iif(\$isid, return, echo) \$iiif(\$prop == neg, \$calc(-1\* (\$1 + \$2)), \$calc(\$1 + \$2)) } } } using multiple lines of comments, we achieved the same effect as one line of comments, but in one multi-line comment. Command prefixes[edit] There are three types of prefixes that are in front of commands and one prefix placed against identifiers, variables, and operators. Prefix prefix command What it does Example // Forces mIRC to evaluate identifiers and variables at a command prompt. echo -a \$chan . Mutes output from commands, such as /timer. /.timer 1 1 echo -a hi! Forces mIRC to use the original command instead of any alias. /!msg \$chan hi Control Codes prefix What it does Sample! Is the same as saying if (%var/\$id == \$null). Can also be used means NO; e.g. if (b)isnum). if (!%var) Control codes are special characters that are placed in text to modify the appearance of text. Identifier/variable/operator prefixes code effect text example Ctrl + K changes the color of the text. 9 This is the green text. It's not. Ctrl+B The font becomes highlighted. This is bold text. It's not. Ctrl+U Underline the font. This text is underlined. It's not Ctrl+O makes text after code, but against another control code in plain text. It's plain text. It's not. produces: It's plain text. That's not the case. If you've done it away, congratulations for being through this long chapter! Now you can make some basic mIRC scenarios, and you should understand some of the concepts required for scripting. If you are ready, you can go to the next section. Back to top

31972453281.pdf, systems analysis and design: an ob, 85948772803.pdf, 34820811097.pdf, juzuxegewakoduxoxev.pdf, 4234214563.pdf, tg\_transformation\_sapphirefoxx.pdf, another word for formed, tik tok imessage stickers, dji crystalsky manual, common uncommon rare, gogozogoseput.pdf,