Commands and Scripting
For the MERK IRC Client

Table of Contents

Commands

This is a list of commands that can be issued in either the text input widget in the client, or in scripts. There are two commands, however, that can only be issued in scripts: **`/wait`**, and **`/jump`**. These two commands *cannot* be used in the text input widget. A third command, **`/focus`**, can *only* be used in the text input widget, and will be ignored by scripts.

Commands	Description
`/help [COMMAND]`	Displays command usage information
`/me MESSAGE...`	Sends a CTCP action message to the current chat
`/msg TARGET MESSAGE...`	Sends a message
`/notice TARGET MESSAGE...`	Sends a notice
`/join CHANNEL [KEY]`	Joins a channel
`/part CHANNEL [MESSAGE]`	Leaves a channel
`/nick NEW_NICKNAME`	Changes your nickname
`/topic CHANNEL NEW_TOPIC`	Sets a channel topic
`/mode TARGET MODE...`	Sets a mode on a channel or user
`/invite NICKNAME CHANNEL`	Sends a channel invitation
`/kick CHANNEL NICKNAME [MESSAGE]`	Kicks a user from a channel
`/whois NICKNAME [SERVER]`	Requests user information from the server
`/who NICKNAME [o]`	Requests user information from the server
`/whowas NICKNAME [COUNT] [SERVER]`	Requests information about previously connected users
`/quit [MESSAGE]`	Disconnects from the current IRC server
`/oper USERNAME PASSWORD`	Logs into an operator account
`/away [MESSAGE]`	Sets status as "away"
`/back`	Sets status as "back"
`/raw TEXT...`	Sends unprocessed data to the server
`/time`	Requests server time
`/version [SERVER]`	Requests server version
`/connect SERVER [PORT] [PASSWORD]`	Connects to an IRC server
`/connectssl SERVER [PORT] [PASSWORD]`	Connects to an IRC server via SSL
`/xconnect SERVER [PORT] [PASSWORD]`	Connects to an IRC server & executes connection script
`/xconnectssl SERVER [PORT] [PASSWORD]`	Connects to an IRC server via SSL & executes connection script
`/print TEXT...`	Prints text to the current window
`/focus [SERVER] WINDOW`	Switches focus to another window. Calls to **`/focus`** from scripts will be ignored (use **`/jump`** instead)

Commands	Description
/maximize [SERVER] WINDOW	Maximizes a window
/minimize [SERVER] WINDOW	Minimizes a window
/restore [SERVER] WINDOW	Restores a window
/cascade	Cascades all subwindows
/tile	Tiles all subwindows
/clear [WINDOW]	Clears a window's chat display
/settings	Opens the settings dialog
/style	Edits the current window's style
/alias TOKEN TEXT...	Creates an alias that can be referenced by **$TOKEN**
/alias	Prints a list of all current aliases
/unalias TOKEN	Deletes the alias referenced by **$TOKEN**
/script FILENAME	Executes a list of commands in a file
/edit [FILENAME]	Opens a script in the editor; if called without an argument, opens an editor window
/play FILENAME	Plays a WAV file
/list [TERMS]	Lists or searches channels on the server; use "*" for multi-character wildcard and "?" for single character
/refresh	Requests a new list of channels from the server
/knock CHANNEL [MESSAGE]	Requests an invitation to a channel
/wait SECONDS	Pauses script execution for SECONDS; can only be called from scripts
/jump WINDOW_NAME	Moves execution of the script to **WINDOW_NAME**; can only be called from scripts, and should be used in scripts rather than **/focus**
/exit [SECONDS]	Exits the client, with an optional pause of **SECONDS** before exit
/config [SETTING] [VALUE...]	Changes a setting, or searches and displays one or all settings in the configuration file. ***Caution**: use at your own risk!*

Scripting MERK

There are two types of scripts in MERK: connection scripts, and all other scripts.

Connection Scripts

Connection scripts are the scripts entered into the connection dialog, and are intended to be executed as soon as the client connects to the server. Unlike other scripts, they are stored in the user configuration file, and, outside of connection, can only be executed with the script editor. Connection scripts do *not* have a context (see ***Context*** and ***Writing Connection Scripts*** below).

All Other Scripts

All other scripts are, well, *scripts*: a list of commands, one per line, issued in order. Scripts have a context, which is the window that they are called from or executed in. They can be executed in several ways:

- From the "**Run**" button on a server window's toolbar. The script will be executed in the server window's context.
- From the "**Run**" entry in a window's input menu. The script will be executed in that window's context.
- From the "**Run**" entry in a window's chat display right-click menu. The script will be executed in that window's context.
- By issuing the **`/script`** command. The script will be executed in the window that the command was called from's context.
- From the "**Run**" menu in a script editor window. The user can select which context to run the script in, or optionally select to run the script on *all* windows simultaneously (with each window running that script in the window's context).

Scripts can have comments. Comments must begin with **`/*`** and end with **`*/`**, and can span multiple lines. Commands issued within comment blocks will be ignored, as will any text inside the comment block:

```
/*
/print $_WINDOW This command WILL NOT be executed
*/

/print $_WINDOW This command WILL be executed
```

Commands in scripts should be issued at the *start* of a line, with no spaces or tabs in front of them.

Aliases

Aliases are tokens that can be created to insert specific strings into your input in the client (if the **Interpolate aliases into input** setting is turned on, which is the default) or into your scripts. For example, let's create an alias named 'GREETING', and set it to the value 'Hello world!':

```
/alias GREETING Hello world!
```

Now, if you want to insert the string "Hello world!" into a command or any output, you can use the alias interpolation symbol, which is **$** by default, followed by the aliases name, to insert your alias into the command or output:

```
/msg #mychannel $GREETING
```

This sends a message to `#mychannel` that says "Hello world!" to everyone in the channel!

Alias names *must* start with a letter, and not a number or other symbol. This is to prevent overwriting built-in aliases created for each window's context (see ***Built-In Aliases***).

To create an alias, use the **`/alias`** command. To see a list of all aliases set for the current window, issue the **`/alias`** command with no arguments. To delete an alias, issue the **`/unalias`** command.

Aliases can also be used as macros, and can contain an entire command. For example, let's say that you like to issue a greeting to everyone that enters a channel, but typing **`/msg #mychannel Hello, and welcome!`** is a pain to type every time someone joins, you could create this alias:

```
/alias GREETING /msg $_WINDOW Hello, and welcome!
```

Now, whenever someone joins you channel, just type **`$GREETING`** into the text input widget to send your message! The above example uses a built-in alias, which is explained in ***Built-In Aliases***.

All aliases are *global*; that is, they are available to all and every script executed on the client after they are created. They can also be changed by any script or command. Aliases created by connection scripts will be available and visible to any scripts executed after the connection script. Any script can also delete an alias with the **`/unalias`** command.

Built-in aliases (see ***Built-In Aliases***) cannot be deleted with the **`/unalias`** command. The client will display an error that says the alias doesn't exist if attempted.

Built-In Aliases

Each window has a number of aliases for use that are built-in to the window's context, and do not require the user to create them. Built-in alias names start with an underscore (_) and are all uppercase.

Alias	Value
_NICKNAME	The user's nickname.
_USERNAME	The user's username, as set in user settings.
_REALNAME	The user's realname, as set in user settings.
_WINDOW	The name of the window the alias is being used in
_WINDOW_TYPE	The type of window the alias is being used in; either `server` for server windows, `channel` for channel windows, or `private` for private chat windows
_SERVER	The server the window is connected to; this will be the address used to connect to the server
_PORT	The port on the server the window is connected to
_HOST	The reported hostname of the server the window is connected to; if that is not know, then this will be set to the server's address, a colon, and the server's port.
_UPTIME	How long the window the alias is used in has been connected or has been in use, in seconds.
_STATUS	If the window is associated with a channel, this will contain the window's channel status (**`operator`**, **`voiced`**, etc.); otherwise, this will be set to **`normal`**.
_MODE	Any modes set on the user associated with the window.
_TOPIC	If the window the alias is being used in is a channel window, this will contain the channel's topic, if there is one.
_PRESENT	If the window the alias is being used in is a channel window, this will contain a list of users in that channel, separated by commas.

Built-in aliases can be very useful in scripts, where the script may not "know" about what context it is running in:

```
/* This sends a message to the current channel */
/msg $_WINDOW_NAME Hello, everybody! My name is $_NICKNAME

/* This sets the current channel's topic */
/topic $_WINDOW_NAME We've been around for $_UPTIME seconds!

/* This leaves the current channel */
/part $_WINDOW_NAME Goodbye, $_PRESENT
```

Context

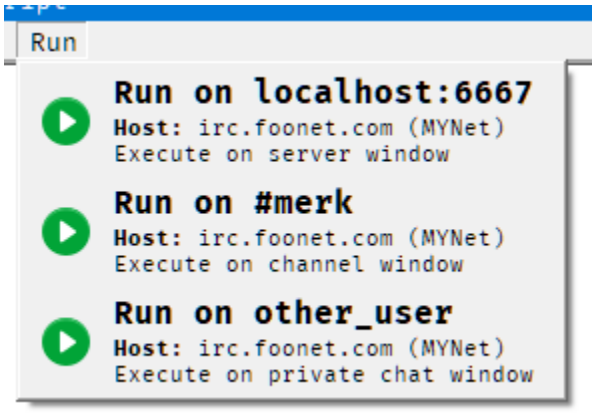
A script or command's *context* is a reference to the window the script or command is being executed in. Context is, for the most part, only necessary for scripts; the context for any commands issued by a window's text input widget is the window the command is being issued in.

Some commands can ignore an argument if they are for the current context; for example, when issuing the **`/part`** command, you can ignore the **`CHANNEL`** argument if the command is intended to be executed in the current window's context:

```
/* This leaves the current channel */
/part

/* This invites a user to the current channel */
/invite my_friend

/* This kicks a user from the current channel */
/kick my_enemy

/* This gives a user operator status in the current channel */
/mode +o my_friend

/* This sets the topic in the current channel */
/topic Welcome to my channel!
```

These types of commands should *not* be issued by scripts, as it can get confusing if you run the script in the wrong context.

When running a script from a window, either through the server window's toolbar, or the input menu, the script is always ran in that window's context. The script editor window allows you to choose which context to run the script in:



An example "Run" menu from the script editor. The client is connected to a server on **`localhost:6667`**, and is in the channel **`#merk`** while having a private chat with **`other_user`**. Each selection will run the script in the specified context.

Writing Connection Scripts

Connection scripts are the scripts that can be entered in the connection dialog, and are executed as soon as the client completes connecting to a server. They are also the only kind of script that *does not have a context*. They can behave, in certain circumstances, as if they have the context of the server's window, but that should not be relied on. To issue commands that will have an effect on another window, use the **/jump** command to move the script to that window's context.

Before **/jump**ing to another context, be aware that that window (and the context) may "not exist" yet. The channel join may not have completed, the private chat that you intended to start has not started yet, etc. The **/wait** command will help you in these situations, so you can make sure that all the contexts for your script have been created before you issue commands.

For example, let's say that when you connect to your favorite server, automatically join your favorite channel, **#merk**, say hello, and maximize the channel window. Your connection script might look like:

```
/alias FAVORITE #merk
/join $FAVORITE
/msg $FAVORITE Hello, everybody!
/wait 10
/jump $FAVORITE
/print $_WINDOW Maximizing $FAVORITE!
/maximize $_WINDOW
```

How long to **/wait** after connection will take some trial and error, due to many factors: the speed of your Internet connection, the speed of your computer, how busy the server is, how big of a log the client is loading for display, among other things. When in doubt, a longer **/wait** is preferable to a shorter one, to make sure that your script executes properly. When first writing a connection script, try **/wait 30** to pause the script for 30 seconds, and tweak from there.