

# Quake 2 Console Tutorial

---

## Abstract

*Author:* JakFrost

*Email:* jakfrost(at)planetquake(dot)com

*Web Site:* <http://www.planetquake.com/console/>

*Created:* January 23, 1998

*Last Modified:* May 18, 2000

*Based On:* Quake 2 v3.12

*Source:* [http://www.planetquake.com/console/tutorials/quake\\_2.html](http://www.planetquake.com/console/tutorials/quake_2.html)

Copyright (C) 1998-2000 JakFrost, All Rights Reserved

This document is a tutorial for the Quake 2 console. It describes all aspects of the console; such as the syntax, the usage of commands and variables, the creation of aliases and bindings, and script. There is a great amount of information contained in this tutorial which should help anybody who is willing to read it and learn from it.

---

## Table Of Contents

1. [Introduction](#)
  1. [Foreword](#)
  2. [Document Conventions](#)
  3. [Definitions](#)
2. [The Console](#)
  1. [Introduction](#)
  2. [Syntax](#)
  3. [Special Characters](#)
    1. [Dollar Sign](#)
    2. [Double-Quote](#)
    3. [Double-Slash](#)
    4. [Newline](#)
    5. [Semicolon](#)
    6. [Space](#)
3. [Console Commands](#)
  1. [Introduction](#)
  2. [Commands](#)
    1. [Action](#)
    2. [Function](#)

3. [Operation](#)
  3. [Variables](#)
    1. [Bitmap](#)
    2. [Command Line Parameter](#)
    3. [Register](#)
    4. [String](#)
    5. [Toggle](#)
  4. [Bindings](#)
    1. [Introduction](#)
    2. [Single Command](#)
    3. [Multiple Commands](#)
  5. [Aliases](#)
    1. [Introduction](#)
    2. [Simple Aliases](#)
    3. [Advanced Aliases](#)
      1. [Action](#)
      2. [Toggle](#)
  6. [Scripts](#)
    1. [Introduction](#)
    2. [The Outside](#)
      1. [Idea](#)
      2. [Execution](#)
      3. [The Format](#)
    3. [The Inside](#)
      1. [Bindings](#)
      2. [Aliases](#)
      3. [Settings](#)
      4. [Documentation](#)
  7. [Afterword](#)
  8. [Version Information](#)
  9. [Legal](#)
    1. [Copyright](#)
    2. [License](#)
    3. [Trademarks and Servicemarks](#)
    4. [Warranty Disclaimer](#)
- 

# 1. Introduction

## 1.1. Foreword

I have decided to write this document to better educate the Quake 2 players about the console and its powerful use. The console is very easy to use and very easy to understand. Now all of those Quake 2 players who have wondered about

aliases and bindings will have a clear picture of what everything means. After reading and understanding this document the console will lose its mystic value and will become just another part of Quake 2.

This is actually my second console tutorial. I have written one before this for the Quake console. That tutorial did explain a lot of things about the console but it did not go into such detail as this tutorial does. That tutorial was like a beta version of this tutorial to tell you the truth. By the way, the Quake console is almost identical to the Quake 2 console, so even though this is a Quake 2 tutorial and I'm using Quake 2 specific commands all of the same ideas can be applied to Quake.

I would like to recommend that while reading this document you take a look at the Quake 2 Console Commands document which is available at this site. That document will become an invaluable tool for you in the understanding of the console. All of the console commands and console variables are described in that document. With this tutorial and that document you will be making aliases and scripts in no time.

I hope you enjoy this tutorial and that it increases your knowledge of Quake 2 in some way. Enjoy...

## 1.2. Document Conventions

Right now I would like to touch-up on a couple of conventions that I use though this document. It is very important to understand exactly what these conventions. Don't worry these conventions are very similar to other conventions used in technical documents, so if you ever read a technical document before you will feel right at home. In case you are wondering, I took these conventions right out of the definitions that are given for them in the HTML Recommendations for the specific markup tags.

### *Code*

I use the convention of code to designate the names of commands or words which appear exactly in the console. Anything that is designated as code stays the same, which is the opposite of variable which would change. For example in the syntax of a command the word "command" is designated as the code since it stays the same throughout the console. The example of the syntax for a command would be `'command (parameter)'`.

### *Keyboard*

I will sometimes use the convention of keyboard input to designate the name of a key that should be pressed. For example I would say something like "...and bind that command to the `pgup` key." I will usually name the key the same as it is named by Quake 2.

### *Quotation*

If I want to show an example inside a paragraph I will use single-quotes ( ' ) to quote the whole example. For example, I would quote the example such as `'command (parameter)'` to show syntax information, or `'bind mouse1 +attack'` to show the exact string of code. The reason why I quote whole lines of code is to make it easier to differentiate between the text in a paragraph and the example itself.

### *Sample*

I will usually use the convention of sample code to show off whole strings of commands which are used in the console. When something is designated as a sample, it should be used exactly as shown. An example of a sample code would be `'bind mouse1 +attack'`. This is somewhat similar to my use of the code convention but they are not exactly the same.

### *Variable*

Variables are parameters which change from one use of a command to another. The command stays the same, it's the variables that change. For example in the example of the syntax of a command the word "parameter" is designated as a variable since it can be anything, a word, a number, etc. The example of the syntax for a command would be `'command (parameter)'`.

Throughout this document you might see examples of code. Most of the code that I show in this tutorial is taken right from my very own Quake 2 script. You will probably see a lot of familiar things along the way. I chose to use examples from my script because it is better to use examples from the real world than some made up ones. Soon after you will learn a lot of things from my script and will become familiar with the way that I play. Consider this an extra benefit like a extra tutorial on a comfortable playing style. :)

## 1.3. Definitions

Below are a couple of confusing terms which I decided to describe. This is just a precaution to make sure that I don't mislead anybody with weird words that I make up and use.

### *command*

A command is bridge to a function which is carried out by the game. The player uses commands to interact with the game through the console.

### *console command*

A console command is a regular command which takes one or more parameters. A console command is just a name of the group of commands to go along with the name "console variables".

### *console variable*

A console variable is a command which holds an alphanumeric value. I might refer to console variables as commands, since they act the same as console commands.

### *executable string*

An executable string is a single line of code which is executed by the console. The ending of an executable string is a carriage-return or a newline which is produced by the ENTER key. An executable string might be composed of a command, its parameters, and special characters.

### *parameter*

A parameter is a value which is used with a command to execute a specific function. Some commands accept parameters which change the function of the command to carry out a different task.

## 2. The Console

### 2.1. Introduction

The console is a wonderful idea for a game as complex as Quake 2. It allows the player to have a lot of control and more chances to customize the game to his liking. It's no wonder that the console was implemented in this game; it would be quite hard to make a menu system to access all the different features that this game offers. The console is a very powerful tool and is very useful to the player.

First off, the easiest way to access the console is to hit the ~ (tilde) key which is located to the left of the 1 key on and above the TAB key on the keyboard. It is very easy to access the console, and you can do it even if you are inside the game playing on a map. All of the features of Quake 2 are accessed and set with the console or with the menu if you don't know the appropriate commands.

It is also possible to access the console from outside the game by using the operating system command line to pass commands to the console before the game is loaded up. The way that this works is that you specify parameters after the name of the executable file to run Quake 2. The name of the console command has to be preceded by the + character to indicate that it is a console command. An example of this would look like `'quake2.exe +exec myscript.cfg`

```
+map base1'.
```

## 2.2. Syntax

First, what "syntax" means is how commands and parameters are put together to form acceptable strings. There are certain rules to follow when putting in commands together with their parameters for the console to understand. The syntax for the Quake 2 console is very simple so don't worry about remembering a lot of weird things. It's all logical.

Any executable string for the console is made up of a couple of parts usually. The main part is the command. Sometimes the command is accompanied by parameters for that command. Sometimes there are even multiple commands in an executable string. A simple executable string might be 'alias' which is just composed of the single `alias` command. A more complex string might be 'alias foormsg "say I am the Foo Master! "', which is composed of the `alias` command and its parameters. Another example of a complex executable string might be 'set name JakFrost;set skin male/howitzer', which is composed of two instances of the `set` command and some parameters.

As you notice each of the above mentioned examples of executable strings always has the command. Sometimes the command has parameters, and sometimes there are more than one command on a single line. Basically you can divide an executable string into commands, parameters, and special characters. As you noticed the second example used quotes in the executable string, and the third example used the semicolon ( ; ). These are special characters to the console and they have special meanings. The sections below will describe exactly what these characters do and how to use them.

## 2.3. Special Characters

### 2.3.1 Dollar Sign

The dollar sign is the \$ character. This character is special in a sense that it allows for the substitution of a variable name preceded with the dollar sign character into the value which is held by that variable. The power of this is that you can display the values that variables hold and you can even copy the values from one variable to another, and back. This allows for an alias to copy the original value of a variable into a backup variable, change the value of the original variable to execute the alias, and after it is finished the original value from the backup variable is copied into the original variable. This way, the alias will return the variable to its original state.

Before this feature was implemented, all of the aliases which had to modify variables would change the variable to a preset value that was stored in the alias. This was problematic in that the player had to change every value in the alias so that those values would be put into the variables after the alias finished executing. If a player set a variable to a setting which was different than the one that appeared in the alias, then after the alias was executed the changes to that variable would be lost because the alias would set the variable into the preset value stored inside the alias.

Also, I would like to mention something about creating backup variables to store values from the original variables. I personally create a backup variable with the same name as the original variable, except that I append the character - to the end of the backup variable. So, if I made a backup variable of the name `variable`, the backup variable would be `variable-`. By using this technique you can be sure of what the backup variable is holding. I think that this is a much better solution, than creating backup variables with extraneous names like `temp`, `backup`, or `a`. This is just a little thing I picked up while learning about Linux. :)

```
echo $name
echo $sensitivity
```

In the examples above, the echo command displays the value of the name and sensitivity variables.

```
set myname $name
```

In the example above the myname variable is created with the copied value from the name variable.

```
set coolname "$name is the Foo Master"
echo $name is the Foo Master
```

In the example above the coolname variable is created which takes the value from the name variable and appends the string " is the Foo Master" to it. In my case the value of the coolname variable would be "JakFrost is the Foo Master". The second example just displays the string "JakFrost is the Foo Master". These examples show that the displayed values can appear inside strings or strings can be appended to the values.

```
// Rocket Jump Quake 2 Alias - Custom Version
alias +rj "rj1;rj2"
alias rj1 "set cl_pitchspeed- $cl_pitchspeed;cl_pitchspeed 100000;wait;+lookdown;
wait;-lookdown;set cl_pitchspeed $cl_pitchspeed-"
alias rj2 "set hand- $hand;hand 2;+moveup;+attack;echo Rocket Jump"
alias -rj "-attack;-moveup;set hand $hand-;centerview"
```

The example above is a version of the Rocket Jump alias which works no matter what is the setting of the hand variable. The alias above will save the value in a backup hand- variable, it will then set the value of hand to 0 so that the rocket is fired from the center of the body. After the alias is finished, the value from the backup hand- variable will be restored into the hand variable. This allows the player to hold the gun in any hand that he wishes, and still make this alias work. The same principle is used with setting the value of the cl\_pitchspeed variable to allow instantaneous look down speed.

### 2.3.2. Double-Quote

The double-quote is the " character. This pair of special characters is used to reverse the function of the special characters, space, and semicolon. Unlike the space and the semicolon which are used to separate, the double-quote is used to joining. Usually when a command accepts only a single parameter but you must include a space or a semicolon in that parameter you would use double-quotes to enclose that string to form a single parameter. Examples of such commands are the alias and name commands which will be discussed later.

```
name "The Foo Master!"
```

In the example above the double-quotes were used to pass the name which is composed of three words separated by spaces as a single parameter to the name command.

```
alias +grapple "use grapple;+attack;echo Quick Grapple"
```

In the example above the double-quotes were used to make the string of commands into a single parameter which will be used by the alias command. As you notice, the string contains spaces and semicolons.

### 2.3.3. Double-Slash

The double-slash is the `//` character pair. This pair of characters signifies a comment. It is usually found inside script files where authors chose to leave comments on their work. Anything after the pair of characters is ignored as a comment and is not executed. It is possible to include a comment on the same line that an executable string is on but I recommend that comments should appear on a separate line above the code to be commented. It is also possible for the comments to appear on the same line as the executable string but I don't recommend that practice.

```
// Quick Grapple Quake 2 Alias
alias +grapple "use grapple;+attack;echo Quick Grapple"
alias -grapple "-attack"
```

In the example above, the double-slash was used to comment on the name of the alias that was used and also on the name of the author for that alias. This is one of the aliases which I use in my script.

```
// These are the user settings that I use when I play the game.
set msg 1
set name JakFrost
set skin male/howitzer
```

In the example above, the double-slash was used to make a comment on what the commands below are used for.

### 2.3.4. Newline

The newline character is the carriage-return character and is invisible. It is invoked by the `ENTER` key. The importance of this character is that it separates commands, just like the semicolon separates commands. Even though this might seem like a very basic concept, I felt that it was important to mention this.

```
set m_pitch -0.022
set sensitivity 17
```

In the example above the two instances of the `set` command are separated by the invisible newline character. If the newline character was missing, the commands would not execute properly.

### 2.3.5. Semicolon

The semicolon is the `;` character. The semicolon character is a very important one. It allows the grouping of multiple commands on a single line. This is a great feature which is very useful when making complex aliases or bindings. The semicolon will become a very useful feature once you start creating complex scripts.

```
alias +sz "set fov- $fov;fov 20;m_pitch -0.004;m_yaw 0.004"
alias -sz "set fov $fov;m_pitch -0.022;m_yaw 0.022"
```

In the example above, you will notice that the semicolon was used to separate three different commands inside the Sniper Zoom alias.

### 2.3.6. Space

The space is the invisible separator character. The space character is another separation character. This character is

similar to the newline character but instead the space character separates commands from their parameters and parameters from other parameters. The space character is also used to separate words when using strings for parameters.

```
set cl_run 1
```

In the example above, you will notice that the `set` command was separated from its first parameter by a space, and the first parameter was separated from the second parameter by another space.

---

## 3. Console Commands

### 3.1. Introduction

Console commands are the commands which are used to carry out certain tasks with the console. There are two major classes of console commands. The first class is the commands which are basically all the left over commands from the variables class. The variables class is second major class of console commands. The difference between a console command and a console variable is that a console command performs a certain task where the console variable is only used to display or store a value.

There are also sub-classes for console commands and variables. These sub-classes are all extraneous to the game, which means that I just use them to describe the syntax for a specific command or a variable and that the game doesn't know anything about these different classes. I felt that it was necessary to classify commands since it is much easier to explain the usage of a command or a variable if you group the appropriate commands and variables. Some of these classes might be merged but I felt that it was necessary to have these very separate groups to avoid any confusion.

A lot of this information was taken directly from the Quake 2 Console Commands document which has a section devoted to grouping and identifying the different classes of commands and variables.

### 3.2. Commands

#### 3.2.1. Action

An action command performs an action when the `+action` part of the command is executed and terminates that action when the `-action` part of the command is executed. When an action command is bound to a key, the `+action` is executed when the key is pressed, and the `-action` is executed when the key is released. The default syntax for an action is `+command` or `-command`.

It is possible for you to make your own action commands by creating two aliases, one starting with a `+` and another one with `-`. The beauty of action commands is that their use is only momentary, so you can create special aliases or functions which only work for as long as a key is held down.

#### 3.2.2. Function

A function command performs a single function based on the parameters included for that operation. A function usually has extra parameters and they are usually necessary for the proper execution. The default syntax for a function is `'command (parameter)'`. All function commands have their own unique syntax.

### 3.2.3. Operation

An operation command performs a single function in the game every time the command is executed. It does not use any parameters. The default syntax for an operation is 'command'.

An operation is exactly the same as a function, except that it does not accept any parameters at all. I felt that it was important to make this group because it is important to know if a command accepts parameters or not. I would be pretty funny if somebody was trying to pass parameters onto an operation command and then get angry that it's not working because he didn't know that it does not accept any parameters.

## 3.3. Variables

### 3.3.1. Bitmap

A bitmap variable is able to toggle more than one feature by using a single value. The way that a bitmap variable works is that a value is assigned to each bit. Each bit is like a toggle, only being able to be set to an off and on value. To turn on the desired features all the values representing individual bits are added together. The default syntax for a bitmap variable is 'variable (value)'. Allowable values for a bitmap variable include integers only.

A bitmap variable might be the toughest one to explain of all of them since it involves a little knowledge of programming and the binary base system. I'll try to do my best to explain this, since it's important to understand it.

Ok, let's take the variable `dmflags` for example. As you might or might not know, it allows the setting of all the different features for deathmatch, such as if weapons disappear and respawn, if health is available on the level, or if weapons have infinite ammo. So, one command is able to set all of these features independently of each other with only one value.

I really don't want to get into the binary part of the explanation here, so I'm going to avoid that topic. For example, let's say that you wanted to create a total blood bath game and wanted to use the settings of: no health, no powerups, weapons stay, spawn farthest, force respawn, no armor, and infinite ammo. You would set a value of 11783 because  $1 + 2 + 4 + 512 + 1024 + 2048 + 8192 = 11783$ . The reason for this value is that each setting represents a value, and to achieve all of the settings you have to add all the values. If for example you wanted to create another game of deathmatch which is not as bloody you might choose the settings of: weapons stay, instant powerups, and spawn farthest. You would set the value of 532 because  $4 + 16 + 512 = 532$ .

Below is a list of the settings to the `dmflags` variable. This list will make it easier to understand the explanation.

- 1 - No Health.
- 2 - No Powerups.
- 4 - Weapons Stay.
- 8 - No Falling Damage.
- 16 - Instant Powerups.
- 32 - Same Map.
- 64 - Teams by Skin.
- 128 - Teams by Model.
- 512 - Spawn Farthest.
- 1024 - Force Respawn.

2048 - No Armor.  
 4096 - Allow Exit.  
 8192 - Infinite Ammo.

The reason why all of this works is because each of the values represents a single bit. When you use a setting with that value in it, that bit is turned on. The bit representation for the value of 532 is 1000010100. If you notice, for that value I chose to only turn on 3 settings thus you have three instances of the number one. I chose to use the setting of "weapons stay" which has the value of 4 and from the list above is the third one from the top. If you notice the third bit from the right is turned on. The same rule applies to the other two settings. By now, I hope that you have seen that each bit has a specific value associated with it. For example the value of 8 determines the value of bit number 4.

### 3.3.2. Command Line Parameters

A command line parameter is a variable which is set from a command line. The reason for setting a variable from the command line is that some variables are write protected once the game starts and cannot be changed. Command line parameters are usually used to enable or disable a handful of features before loading up the game. The default syntax is "quake2.exe +set variable (*value*)".

This variable has replaced the old use of the command line parameters which started with a dash -parameter. This is an advantage because you can check inside the game if a command line parameter has been accepted correctly or not. The old use of command line parameters was limited in that the user had to guess if the value has been accepted or not.

### 3.3.3. Register

A register variable is able to store any type of numeric information. Register variables are used for numeric data which does not have clearly defined boundaries. The default syntax for a register variable is "variable (*value*)". Allowable values for a register variable include integers, fractions, and negative values.

This group of variables is similar to the command group of functions in that all the left over variables are put into the group. Registers usually hold numeric information, sometimes positive numbers, negative numbers, whole number, or fractions.

### 3.3.4. String

A string variable is able to store any type of text information. The default syntax for a string variable is 'variable "(*text*)"'. Allowable values for a string variable include all numbers and all text characters.

The reason for string variables is to allow the storing of human readable information, such as names, addresses, or other things. It should be clear to see why string variables exist in the game.

### 3.3.5. Toggle

A toggle variable is able to turn a feature on, off, or set to an alternative setting. All the values for a toggle are integers, usually only 0 and 1. Sometimes there are alternative settings for a toggle, in which case subsequent numbers are used. The default syntax is 'variable (*value*)'. Allowable values for a toggle variable include integers only in increments of 1.

Toggle variables are a little similar to bitmap variables but toggle variables are only able to store the information about

one setting instead of multiple settings.

---

## 4. Bindings

### 4.1. Introduction

Bindings allow the connection of commands to keys. The binding of keys is done with the use of the `bind` command. This command allows the player to bind a command or a series of commands to a key or a button. This is very useful for a number of reasons. Because of this command, you will be able to hit a key and have a single command executed or a whole bunch of commands. This command also allows the input devices such as the keyboard and mouse to serve as an interface to the console.

### 4.2. Single Command

A single command binding is the most simple of all bindings. It just binds a single command to a key. All of the action commands are bound with the use of a single command binding.

In the example below you see the bindings of all the movement keys which I use. You will notice that the binding is composed of the `bind` command, the key to bind such as `e`, and the command which should be bound to the key which would be `+forward`.

```
bind e +forward
bind s +moveleft
bind d +back
bind f +moveright
bind alt +movedown
bind space +moveup
```

In the example below you will notice that the keys `F7` and `F8` were bound to non-action commands. Just another simple example.

```
bind f7 menu_keys
bind f8 menu_video
```

### 4.3. Multiple Commands

The binding of multiple commands is somewhat more advanced. It allows one key to perform multiple functions. The benefit of this is that it allows the creation of key-stroke saving keys, where you press one key and it executes a frequently used list of commands.

In the example below, you will notice that three commands have been bound to the keys. Each command was separated by the semicolon character which is used especially for separating commands. One thing to remember is that when binding multiple commands you have to enclose the commands in double-quotes since the semicolon character and spaces were used.

```
bind F6 "echo Quick Saving...; wait; save quick"  
bind F9 "echo Quick Loading...; wait; load quick"
```

---

## 5. Aliases

### 5.1. Introduction

Aliases is where the real power of the console lies. With aliases you can create very elaborate strings of commands to perform almost any function imaginable. Aliases kind of imitate commands in that after an alias is created you can execute the commands for that alias just by specifying the name of the alias. The only real draw back that aliases have is that they can't accept parameters like real commands do.

It is possible to create an alias which will execute another aliases. For as long as the game is concerned an alias is an operation command. You can create an alias which will execute other aliases and in turn those other aliases execute even more aliases. So as you see, you are not only limited to using aliases for commands or variables, you can create execution trees of aliases. You can even create other aliases from inside aliases.

There are different types of aliases. There are those very simple aliases which are just strings of commands executed by the use of the alias name. Then there are action aliases which mimic the action commands. There are also toggle aliases which kind of mimic the toggle variables. There are also more advanced aliases then those, but they are too complex to try to classify.

### 5.2. Simple Aliases

Simple aliases are just strings of commands condensed into a single command which is the name of the alias. Simple aliases are usually used to save time on frequently used commands, they are also quite similar in syntax and use to the single command bindings.

In the example below is an alias which when executed would execute the commands within it. As you see the string is composed of the `alias` command, the name of the alias which is `settings`, and the following commands which are separated by semicolons.

```
alias settings "set msg 1;set name JakFrost;set skin male/howitzer"
```

### 5.3. Advanced Aliases

There are aliases which I chose to classify as advanced aliases because their functions are not as basic as the simple aliases. There are a bunch of advanced alias groups, and each one differs from another by the function and the type of coding that is used.

#### 5.3.1. Action

Action aliases are just like action commands in all respects. The benefit of using action aliases is for special functions which only should be active when a key is pressed and held down, and when the key is released that function should stop.

In the example below you have the famous Quick Grapple alias. As you notice it is actually two aliases. One starts with the + character and enables the function as long as the key is pressed and held down. The other starts with the - character and disables the function once the key is released.

```
bind x +grapple
alias +grapple "use grapple;+attack;echo Quick Grapple"
alias -grapple "-attack"
```

### 5.3.2. Toggle

Toggle aliases are similar to toggle variables, in that they usually turn a feature on or off. Sometimes though, toggle aliases select from a list of settings which necessarily does not involve the enabling or disabling of a feature. The benefit of using such aliases is that you can choose from a list of options just by using one alias instead of having to resort to using multiple aliases.

In the example below you have an alias which will cycle through all the settings for setting the crosshair. The name of this alias is `ch` which stands for crosshair. The first line of this alias creates the initial setting in that when the `ch` alias is executed it will execute the `ch1` alias. When the `ch1` alias is executed by the use of the `ch` alias it will in turn set the `ch` alias to execute the `ch2` alias next time. Because of this, a cycle is created which allows the single `ch` alias to execute all of the other aliases on subsequent executions. The other part of the other aliases sets the `crosshair` toggle variable to another setting and displays a message to the player.

```
bind x ch
alias ch ch1
alias ch0 "alias ch ch1;crosshair 0;echo Crosshair Off"
alias ch1 "alias ch ch2;crosshair 1;echo Crosshair 1"
alias ch2 "alias ch ch3;crosshair 2;echo Crosshair 2"
alias ch3 "alias ch ch0;crosshair 3;echo Crosshair 3"
```

The example below is very similar to the example above, except that a different coding was used to perform the same function. As you notice the `bind` command was used to achieve the cycling effect. In this alias, every time an alias is executed the X key is rebound to another alias. This type of a toggle alias is not as flexible as the above version in that if the player wants to move the alias to another key, he has to change every instance of the `bind` command instead of just changing just one like in the alias above.

```
bind x ch0
alias ch0 "bind x ch1;crosshair 0;echo Crosshair Off"
alias ch1 "bind x ch2;crosshair 1;echo Crosshair 1"
alias ch2 "bind x ch3;crosshair 2;echo Crosshair 2"
alias ch3 "bind x ch0;crosshair 3;echo Crosshair 3"
```

## 6. Scripts

### 6.1. Introduction

Scripts are files which have been made by players to hold their aliases, bindings, and settings. Scripts are quite useful in that all the information necessary for setting up the controls for the game reside in a single file.

The benefit of that is that you can take that single file to other computers and use it, or distribute the file on the Internet for other people to see. The problem with putting settings in the `config.cfg` file is that it is modified by Quake 2 every time it boots and it does not maintain aliases. Another benefit of scripts is that only the information for setting the bindings and controls are stored in the file so that settings which are specific to the computer are not stored in the file. The reason why this is a benefit is that if you move the script from one computer to another you don't have to worry about sound or video card settings for that computer. It is also easier to edit a script which is cleanly made instead of having to go through all the crud that's inside the `config.cfg` file.

## 6.2. The Outside

### 6.2.1. Idea

Basically before making a script you have to have an idea of what it is and what it does. It is also a good idea to know what purpose the script is supposed to serve. Most of the time scripts are made to be distributed to other computers or given to friends.

The script file should hold information which is only necessary for settings the controls and settings for the player and should not contain any information which is specific to the machine. The reason for this is that it is much easier to distribute scripts like that since the receiving person doesn't have to go through the script file and take out all the instances of code which is computer dependent, such as the video card settings, or the sound settings. Usually, it's a good idea to place all of the necessary bindings, all of the aliases, and a couple of player dependent settings inside the script.

### 6.2.2. Execution

Before you go off and starting making aliases and scripts you should first think where you are going to put all of that work. First, you should think of a filename where your script is stored. It is not a good idea to store your script inside the `autoexec.cfg` in the `quake2/baseq2/` directory because that file should be kept as small and possible only with links to other files which should be executed. Think of a file name, maybe one that is based on the nick name that you use when you play. For example, for my script I chose the filename of `jf_q2cfg.cfg`. Also, make sure that when you make your script file that you place it in the `quake2/baseq2/` directory.

If you ever want to create scripts which are made up of more than one file then you should create your own directory to keep all of those files there and not clutter up the main directory. For example, I would make a directory like `quake2/baseq2/jf_q2cfg/` to hold all the files if I made a large script. It's a good idea to be organized and not clutter up your system, that's what directories are for.

The way that you want to have your script executed by Quake 2 is to put in a line like `'exec jf_q2cfg.cfg'` in the `quake2/baseq2/autoexec.cfg` file. That file is executed by Quake 2 every time that Quake 2 is started up. Remember, avoid putting in other commands into the `autoexec.cfg` file.

Below is an example of my `autoexec.cfg` file. The first two lines of code prevent the startup introduction from playing. The last line executes my script.

```
alias dl ""
```

```
toggleconsole
```

```
exec jf_q2cfg.cfg
```

### 6.2.3. The Format

I would now like to describe a format which I use for making script files. This is just a recommendation from me and is in no way related to how Quake 2 will feel about your script. These are only things which I do to keep my script organized.

Your script should have a heading which would identify it as a Quake 2 script. The reason for having a heading is if somebody stumbles on your script file he will know exactly what that file does. Below is the heading which I have in my script file.

```
// JakFrost's Quake 2 Configuration Script
// =====
// Copyright 1998 JakFrost, All Rights Reserved.
```

Next, your script should have some information about it, like its author, how to contact the author, its version or release, and the last date of its update. It is important to state such things in a file so that it's easy to find out who the script belongs to and who the rightful author is. You might want to create a comment inside the script file which would specify all of that information or just create a small section in your alias with a couple of echo commands which would display that information on the Quake 2 console whenever the game starts up.

Below is the section of my script file which displays the information for my script to the Quake 2 console upon startup.

```
echo
echo ----- Configuration Script -----
echo Version: Release 4
echo Date: February 20, 1998
echo Author: JakFrost
echo -----
echo
```

The example shown below would be if I wanted to include a heading in the script file which would not be displayed on the Quake 2 console. This is for the people who do not want to clutter up the console when the game starts up but still want the information about the script displayed. This information should appear directly under the heading for the script file.

```
// JakFrost's Quake 2 Configuration Script
// =====
// Copyright 1998 JakFrost, All Rights Reserved.
//
// Version: Release 4
// Date: February 20, 1998
```

Also each script should have an ending marker. This is totally not necessary for the script or for Quake 2 but I include it in my script files. The reason why I include this is that Quake and Quake 2 suffer from a couple of bugs, one of which

involves commands which appear at the very end of a script. If a command appears on the last line of the script it will not be executed when the script is executed by Quake 2. If you put a comment on the last line of your script you will be able to avoid this bug.

Below is an example of the ending marker that I use for my script files. The acronym EOF stands for end-of-file.

```
// EOF
```

## 6.3. The Inside

### 6.3.1. Bindings

I think that bindings are the most important part of a script file. After all, bindings are what set up your controls for playing. The reason that you should include your bindings in your script file is that you can avoid the ever changing structure of the `config.cfg` file which is changed every time that Quake 2 starts. Settings bindings in a script also allow you to make the script more modular, enabling for the distribution of the script without having to distribute the `config.cfg` file.

The one thing that you should put in a script before you put in any bindings is the `unbindall` command which will clear all the bindings that Quake 2 has. This is very important because some phantom bindings are saved in the `config.cfg` file which could come back later to haunt you and screw up your setup. You have to remember that you will have to redo all of the bindings after you use the `unbindall` command.

When you put your bindings into your script file, it is a good idea to organize the bindings in groups. Later on you will find that it's easier to change bindings if they are grouped instead of having all of them in an alphabetical list. It would be a good idea to group bindings by the functions that they carry out. For example you could separate some bindings and make them into the "Movement Keys" group, or take all of the number key bindings and make a group called "Weapon Keys". This is not necessary but it's just for being neat.

Below is the bindings section from my script. Look over this list and notice the groupings of bindings by their functions. Also if you are interested you will be able to learn the keys that I use to play. This is just an example to show you what I mean in the paragraphs above.

```
// Bindings
// -----

unbindall

// Special Keys
bind escape togglemenu
bind pause pause
bind ` toggleconsole
bind - sizedown
bind = sizeup
bind tab "cmd help"

// Function Keys
bind f1 "cmd help"
```

```
bind f2 menu_savegame
bind f3 menu_loadgame
bind f4 menu_keys
bind f5 menu_startserver
bind f6 "save quick"
bind f7 menu_playerconfig
bind f8 menu_addressbook
bind f9 "load quick"
bind f10 menu_quit
bind f11 menu_credits
bind f12 screenshot

// Weapon Keys
bind 1 "use blaster"
bind 2 "use shotgun"
bind 3 "use super shotgun"
bind 4 "use machinegun"
bind 5 "use chaingun"
bind 6 "use grenade launcher"
bind 7 "use rocket launcher"
bind 8 "use hyperblaster"
bind 9 "use railgun"
bind 0 "use bfg10k"

// Movement Keys
bind e +forward
bind s +moveleft
bind d +back
bind f +moveright
bind alt +movedown
bind space +moveup

// Quick Keys
bind q "use rocket launcher"
bind w "use hyperblaster"
bind r "use railgun"
bind t "use bfg10k"
bind a qw
bind g +rj
bind z "use power shield"
bind x "use grenades"
bind v "use invulnerability"
bind b "use quad damage"
bind shift +sz
bind ctrl +speed

// Option Keys
bind y "use environment suit"
bind u "use rebreather"
bind i "use silencer"
```

```
// Inventory Keys
bind [ invprev
bind ] invnext
bind \ inven
bind ' invdrop
bind enter invuse

// Wave Keys
bind h "wave 0"
bind j "wave 1"
bind k "wave 2"
bind l "wave 3"
bind semicolon "wave 4"

// Message Keys
bind n messagemode
bind m messagemode2

// Mouse Buttons
bind mouse1 +attack
bind mouse3 +grapple
```

### 6.3.2. Aliases

It is a good idea to place all of your favorite aliases in the script file. That's what a script file was designed to do. Your script is the only place to actually keep aliases since you can't put them inside the `config.cfg` file since they will be deleted by Quake 2.

Before you start putting in all of your little aliases, I think that it would be a good idea to give names to those aliases in case other people like those aliases and use them. I personally name all of the aliases that I use which serve specific functions.

There is one thing that I would like to mention about aliases. Most of the time people take aliases posted on the Internet which were made by other people. I think that those people deserve respect and recognition for the work that they did. I feel that it would be a good idea to include some information about the original author of an alias right on top of that alias and below its name. If the author information is included in an alias and somebody decides to take that alias out of your script and use it in their script, they could just cut and paste the alias and information which would save them time.

Below are all of the aliases from my script file. Notice that all of the aliases have names to them and include information about the author.

```
// Aliases
// -----

// Quick Grapple Quake 2 Alias
alias +grapple "use grapple;+attack;echo Quick Grapple"
alias -grapple "-attack"
```

```
// Quick Weapon Quake 2 Alias - Best Weapon Version
// Author: Craig "Vermin" Morris (cmorris@aol.com)
alias qw "msg 3;qw1;qw2;msg 1;echo Quick Weapon"
alias qw1 "wait;use grenades;wait;use shotgun;wait;use super shotgun;wait;use
grenade launcher;wait;use machinegun;"
alias qw2 "wait;use chaingun;wait;use railgun;wait;use rocket launcher;wait;use
hyperblaster;wait;use bfg10k;"

// Rocket Jump Quake 2 Alias - Custom Version
alias +rj "rj1;rj2"
alias rj1 "set cl_pitchspeed- $cl_pitchspeed;cl_pitchspeed 100000;wait;+lookdown;
wait;-lookdown;set cl_pitchspeed $cl_pitchspeed-"
alias rj2 "set hand- $hand;hand 2;+moveup;+attack;echo Rocket Jump"
alias -rj "-attack;-moveup;set hand $hand-;centerview"

// Sniper Zoom Quake 2 Alias - Inverted Mouse Version
alias +sz "set fov- $fov;fov 20;m_pitch -0.004;m_yaw 0.004"
alias -sz "set fov $fov-;m_pitch -0.022;m_yaw 0.022"
```

### 6.3.3. Settings

You should also include a section where you keep a couple of your settings for your configuration. It is a good idea to place things such as the player's name here, the skin that he uses, and a couple of other settings. This section is just for things that you use to help you with your bindings and aliases.

Below is the settings section from my script. Notice there are a lot of personal settings there. There are also a couple of settings for my keyboard and my movement.

```
// Settings
// -----

// Personal
set hostname "JakFrost's Fridge"
set msg 1
set name JakFrost
set skin male/howitzer

// Other
set cl_run 1
set crosshair 3
set freelook 1
set hand 0
set lookspring 0
set lookstrafe 0
set m_filter 0
set m_pitch -0.022
set skill 2
```

### 6.3.4. Documentation

If you are planning to release your script on the Internet, it would be a good idea to write something about your script file. It is usually a good idea to give some background information about the script such as the author information, the version information, and the date of the last update. You should also mention a couple things about the bindings in the script and maybe explain the aliases that you used. I would recommend that you write something about your script that would enable other people to know what's going on.

Also, if you are planning to release your script on the Internet it would be a good idea to release it with the script file and the documentation file inside a compressed archive file, usually a .zip file. I would recommend that you would name your documentation file with the same file name as your script file except that the extension to that documentation should be .txt. So if my script filename is `jf_q2cfg.cfg` the filename of the documentation file would be `jf_q2cfg.txt`. I would also recommend that the filename of the compressed archive be the same as that of your script and documentation. My script file is usually released on the Internet with the filename of `jf_q2cfg.zip`.

If you plan to include documentation with your script, I suggest you make it a regular text file. If the script is written in plain text I think that the documentation file should be too. Please, avoid releasing your file in any other format since not everybody can read Word documents or Rich Text Files.

---

## 7. Afterword

If you are reading this part I guess that you didn't fall asleep yet. By now if you understood everything mentioned above you should have a pretty clear picture of how the console works and what you can do with it. Make sure that you take a look at that Quake 2 Console Commands document that I mentioned to get all the information about all of those Quake 2 commands and variables which have been used throughout this document. You should also know a little about my script and the setup that I use to play.

If you are interested in obtaining my script and other scripts for Quake 2 please visit Walnut Creek ([ftp.cdrom.com](ftp://ftp.cdrom.com)) by using the link <ftp://ftp.cdrom.com/pub/quake2/console/>.

If you found any problems with this document such as spelling, grammar, or technical errors please contact me so I could fix them. Also, I would like to mention that even though I wrote this alias to help others learn about the console I am not a personal tutor. Please do not send me email asking me to help you with console problems because I will not answer such email.

---

## 8. Version Information

**May 18, 2000**

- Updated the legal and licensing terms of this document and it is now released under the Free Software Foundation's GNU Free Documentation License, Version 1.1.
- Removed the links to the Cascading Style Sheet file which was providing the stylistic markup for this document in order to solve an existing major problem with troublesome web serving software and to promote better accessibility and availability of this document for the future at the price of removing the coloring and formatting

markup from this document.

- Changed the format of my email address from URL format to a proprietary format in order to defeat the automatic gathering of my email address by web crawler software and to prevent the increasingly annoying barrage of unsolicited commercial advertisements that I have been receiving.
- Removed all other HTML LINK tags to prevent future problems with web serving software.
- Renamed the Index section to Table Of Contents to correct this long lasting mistake.

## November 6, 1998

- New legal terms for this document. It is now released under the Free Software Foundation's GNU General Public License Version 2.
- Changed the definition list formatting in sections 1.2. Document Conventions and 1.3. Definitions from bold to italic.

## September 29, 1998

- Updated the HTML markup in the document. This should be the last such update because I think I solved all of the previous problems with HTML and the correct display of this document.
- Updated the HTML 4.0 declaration to include the URI of the DTD, as stated in the standard.
- Added the MIME Content-Type and Content-Language fields to explicitly set the character set encoding and language information.
- Added HTML meta information to each document to include all necessary header information to help search engines.
- Changed the *Modified:* field to *Last Modified:*, and *Location:* to *Source:* to avoid ambiguity.
- Changed the usage of bold and italic stylistic markup in order to facilitate better display of this document in browsers which do not support Cascading Style Sheets.

## July 20, 1998

- I have removed my email address from this document because I was getting unwanted emails. If you want to contact me, go to my web site.

## July 16, 1998

- Changed the internal HTML style for indentations and also to space out the fields for easier reading.
- Added subsection 9.4. Trademarks in the 9. Legal section to take care of any trademark legalities.
- Added a copyright notice to the Abstract section along with a short description about this document.

## March 13, 1998

- Made some formatting changes.

## March 1, 1998

- Fixed a lot of spelling and grammatical errors thank to the careful eye of [David "Veritas" Luntz](#). Without readers like him, my bad grammar and spelling would be free to roam the 'net unchecked. I extend my greatest thanks to David for his proofreading. :)

## February 24, 1998

- Changed the description of "Double-Backslash" to "Double-Slash" thanks to [Martin Kuhne](#) for noticing this huge blunder. I'm surprised that I didn't catch this error a very long time ago when I was writing the Quake Console Tutorial.
- Switched the sections 2.3.2 and 2.3.3 around because of the above mentioned change to keep the sections in alphabetical order.

## February 22, 1998

- Added the dollar sign special character which reappeared in the Quake 2 v3.12 code to the console syntax section.
- Added my email address to the Abstract section. It was been removed long time ago because of too many emails, and I hope that this isn't the case this time.
- Updated the outdated comments in the alias displayed in section [2.3.2](#).
- Changed all of the Quick Hook aliases to Quick Grapple because of the new Quake 2 CTF release.
- Updated the example of my script file to my current settings.
- Added a sentence to each of the special console characters to show what they look like.
- Substituted the old Rocket Jump and Sniper Zoom aliases for newer versions of those aliases which use the dollar sign special character for storing variable values.
- Moved the Abstract section to the top of the document above the Table Of Contents.

## February 5, 1998

- Fixed a few grammatical errors in the document, thanks to [Clambake](#).

## February 2, 1998

- Added the Legal sections to allow free distribution of this tutorial.
- Added the *Location:* field in the Abstract so people can find the original location of this document.

## February 1, 1998

- Fixed two spelling errors thanks to [KingWolf](#).

## January 23, 1998

- Original release.

---

## 9. Legal

### 9.1. Copyright

This work is copyrighted pseudonymously by the author under all applicable laws. The author reserves all rights to this work. The copyright statement for this work is "*Copyright (C) 1998-2000 JakFrost, All Rights Reserved*".

## 9.2. License

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts and no Back-Cover Texts. You can access the license at <http://www.gnu.org/copyleft/fdl.html> or you can write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA to request a copy. Please preserve all authorship and contact information when distributing or modifying this work. All derivative works or works containing parts of this work must be released under the same license or a later version of it.

## 9.3. Trademarks and Servicemarks

All trademarks and servicemarks used in this work are acknowledged and are the property of their rightful owners.

## 9.4. Warranty Disclaimer

THIS WORK IS PROVIDED "AS IS" WITHOUT ANY WARRANTY; WITHOUT EVEN THE IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THE AUTHOR IS NOT LIABLE FOR ANY RESULTS ARISING FROM THE USE OF THIS WORK. ALL RISKS AND RESULTS THEREOF FALL ON THE USER.

---

Copyright (C) 1998-2000 JakFrost, All Rights Reserved