Creating a cfg file.

Before you start learning to script you have to know in what kind of a file you place your script. These files are known as config files or cfg files. Cfg stands for the extension of the file, e.g. weapons.cfg.

You do not need any expensive software to create these files. You can use a text editor like notepad, wordpad or notepad++. Most people use notepad, so I will also be using it in my examples in all the tutorials.

The tricky part is in the saving of the file. We will now create an empty cfg file called test.cfg. Open notepad and just, like you always would when saving a file, go to:
File --> Save as

Now there are 2 ways of saving this file correctly:
1. Change the file type to "All files" and set your file name to weapons.cfg, like you can see in the image below

2. Leave the filetype as "Text Documents (*.txt)" and set your file name as "weapons.cfg". It is important that you place your filename in between the " ", just like in the image below.

You now have an empty cfg file in which you can place a script. Of course you decide where you save the files. In the following thread we will discuss where to place the files, when you want to start using them

---

Creating an autoexec.cfg and using downloaded scripts.

Autoexec

There is one special cfg file and it's called the autoexec file. As you can probably imagine by its name, this file automatically executes itself and thus running all the commands and scripts stored inside. This is why this file should be the basis of all your scripts. From this file you will execute all your other cfg files.

Creating the autoexec.cfg file is done in exactly the same way like any other cfg file, which we discussed in the previous thread. This time saving it as autoexec.cfg.

Using downloaded scripts

To use a script you do not need almost no knowledge about scripting. The only thing you need to know is how to make an autoexec, because all scripts you use can be executed from this file. Let's say you've made an autoexec file and you have downloaded 2 scripts called weapons.cfg and vsays.cfg. First thing you do is open your autoexec.cfg. In this file you have to type the 2 following lines to execute the 2 scripts you have downloaded:

You do not have to be Einstein to figure out that if you execute your autoexec file, your 2 scripts weapons.cfg and vsays.cfg will also be executed and thus your 2 scripts will work.

The next question is where to put the files. Well in most cases all cfg files go in your etmain directory. There are some exception, but those will be dealt with at a later stage.

Now that you have placed all the files in your etmain directory, we will have to execute them. So you start up the game and go onto your favorite server. You then pull down your console, which is done with the tilde key at the top left of your keyboard.

You now execute your autoexec by typing:
/exec autoexec (followed by an ENTER and no need to place the file extension .cfg behind the command).Just like you can see in the following image:

You can now start using your scripts. How to actually make your own scripts is discussed in the following threads.

Using more than one cfg file.

People often ask me why should I use more than one cfg file? Why not only use your autoexec.cfg file? Well there are 2 answers in my opinion:
* The maximum file size and
* Creating clarity and structure in your config.

I'll discuss both!

Maximum file size
The file size of a cfg file cannot exceed 16kb. Since it is no more than a simple text file(, but with another extension), you can add a lot of stuff to it before you reach that maximum.

Clarity and structure
Instead of putting everything in one file it's better if you try to divide all your settings, binds and scripts over different cfg files, instead of putting everything into 1 cfg file. Let's say you put everything into 1 file and after a perioid of time you have produced over 200 lines of code in that file. Wouldn't it be hard to find a certain line of code in all that text? Now if you were searching for a certain script and it was in a seperate cfg file, you would not have a problem finding it.

We talked about creating an autoexec file in the second tutorial and about executing it. All the commands inside will be executed. Now your next question would probably be: "Do I have execute all the scripts in console, like I have to do with the autoexec file, explained in the second tutorial? Well of course all cfg files have to be executed, but not manually in your console. Only your autoexec is executed from your console and all the other cfg files are executed from the autoexec.cfg, also explained in the second tutorial(in the part of using downloaded scripts).

Let me just give you one more simple example. Say you have 3 cfg files:

* settings.cfg
* controls.cfg
* scripts.cfg

You execute these files from your autoexec.cfg with the following lines:
exec settings.cfg
exec controls.cfg
exec scripts.cfg

If you then execute your autoexec.cfg in console, your autoexec will execute the other 3 cfg files. Piece of cake!

Binding.

In order to assign a command to a key, you have to bind a key to that command. This is done as followed:
bind key "command"

For example:
bind x "vsay hi"

The binded key in this example is x and the command is vsay hi (We will discuss the vsay command later in tutorial 5). Now if you would press x in the game, you would say hi.

The quotation marks(" ") are only obligatory if you use multiple commands for 1 bind. So this would also do:
bind key command

If you start binding more than 1 command to 1 key, then it has to be:
bind key "command1; command2"

As you can see all commands are seperated by a semicolon(;).

This is how you bind a key. In the following threads we will discuss all the things you can actually bind to a key.

The say and vsay commands.

In the game of Enemy Territory there are 2 types of chats: say and vsay. Let's discuss them both.

say

The say command is nothing but an ordinary chat. Just like when you type a line of text to say to your friends in the chat area. For example, when you enter your favorite server you might wanna say something like: "Hello everyone!". Instead of typing that every time you enter the server, you could bind a key to that say command.

Let's say you want to use x as your key for this bind, it would look like this:
bind x "say Hello everyone!"

Now every time you press x, you will say "Hello everyone!" in the chatarea.

So when binding a say, you first type the command say followed by what you want to appear in the chatarea.

vsay

There are also recorded voice messages in ET. These are accessible through the vsay menu when you press v. With this menu you can say things like: hi, bye, thank you, you're welcome, etc. These vsay messages can also be binded to a key. In the previous tutorials I gave you the following example:
bind x "vsay hi"

When pressing x you would hear the recorded voice message hi. So when binding a vsay, you first type the command vsay followed by the command for the voice message you want to hear. Of course you have to know what the commands are for these voice messages. So here is the list with all the possible vsay commands. This list first states the voice chat you would hear and behind the - what the command is for this voice chat:

1.Statements
Path cleared. - PathCleared
The enemy is weakened. - EnemyWeak
All clear. - AllClear
Incoming! - Incoming
Fire in the hole! - FireInTheHole
I'm defending. - OnDefense
I'm attacking. - OnOffense
Taking fire! - TakingFire
Mines cleared. - MinesCleared
Enemy in disguise. - EnemyDisguised

2. Requests
Medic! - Medic
I need ammo! - NeedAmmo
I need backup! - NeedBackup
We need an engineer! - NeedEngineer
Cover me! - CoverMe
Hold fire! - HoldFire
Where to? - WhereTo
We need Covert Ops! - NeedOps

3. Commands
Follow me! - FollowMe
Let's go! - LetsGo
Move! - Move
Clear the path! - ClearPath
Defend our objective! - DefendObjective
Disarm the dynamite! - DisarmDynamite
Clear the mines! - ClearMines
Reinforce the offense! - ReinforceOffense
Reinforce the defense! - ReinforceDefense

4. Talk
Yes! - Affirmative
No! - Negative
Thanks a lot! - Thanks
You're welcome. - Welcome

Sorry! - Sorry
Oops! - Oops

5. Global
The enemy is weakened. - EnemyWeak
Hi! - Hi
Bye. - Bye
Great shot! - GreatShot
Yeah! - Cheer
Hold your fire! - HoldFire
Good game! - GoodGame

6. Function
I'm a soldier. - IamSoldier
I'm a medic. - IamMedic
I'm an engineer. - IamEngineer
I'm a field ops. - IamFieldOps
I'm a covert ops. - IamCovertOps

7. Objectives
Command acknowledged! - CommandAcknowledged
Command declined! - CommandDeclined
Command completed! - CommandCompleted
Destroy the primary objective! - DestroyPrimary
Destroy the secondary objective! - DestroySecondary
Destroy the construction! - DestroyConstruction
Construction underway! - ConstructionCommencing
Repair the vehicle! - RepairVehicle
Destroy the vehicle! - DestroyVehicle
Escort the vehicle! - EscortVehicle

8. Fire team chats
Attack! - FTAttack
Fall back! - FTFallBack
Cover me! - FTCoverMe
Disarm the dynamite! - FTDisarmDynamite
Fall back! - FTFallBack
Soldier, covering fire! - FTCoveringFire
Deploy mortar! - FTMortar
Heal the squad! - FTHealSquad
Heal me! - FTHealMe (will show a medic icon over head)
Revive team mate! - FTReviveTeamMate
Revive me! - FTReviveMe (will show a medic icon over head)
Destroy objective! - FTDestroyObjective
Repair objective! - FTRepairObjective
Construct the objective! - FTConstructObjective
Deploy landmines! - FTDeployLandmines

Disarm landmines! - FTDisarmLandmines
Call airstrike! - FTCallAirStrike
Call artillery! - FTCallArtillery
Resupply squad! - FTResupplySquad
Resupply me! - FTResupplyMe (will show a ammo icon over head)
Explore area! - FTExploreArea
Check for land mines! - FTCheckLandMines
Destroy satchel objective! - FTSatchelObjective (same file as FTDestroyObjective)
Infiltrate! - FTInfiltrate
Go undercover! - FTGoUndercover
Provide sniper cover! - FTProvideSniperCover

So one more example. If yu would like to bind x to the vsay Command Acknowledged!, the you bind it as followed:
bind x "vsay CommandAcknowledged"

Teamchat

The say and vsay command are global chats and will be seen/heard by everyone. In some cases you might not want your enemy to see/hear something. This can be avoided by adding _team to your say or vsay command. So the following examples will only be seen/heard by your team:
bind x "vsay_team NeedAmmo"
bind y "say_team The enemy is coming! Fall back!"

Customizing your voice chat

When using a voice chat, you do not only hear it, but also see it in the chat area. If I would use the vsay needAmmo, I would not only hear I need ammo, but would also see this in the chat area:

If you want you could change the part you see in the chat area. Let's say you would want to use the vsay needAmmo, but you want it to say "I need bullets!" in the chat area. This bind is set up like this:
bind x "vsay needAmmo I need bullets!"

Pretty simple you just place the text you want in the chat area behind the vsay command. That's all there is to it. The result:

Using colours for your name and binded chats.

The most common use for colours is to make your gaming name and your binded chats look better. Why say "Hi everyone!", when you can say "Hi Everyone!".

Colour codes

Colours are made with colour codes. A colour code consists of a ^ followed by a letter or number. Each letter or number represents a certain colour. In the following images you can see the number/letter with their corresponding colours:

Binded chats

Let's get back to the example in the previous thread, where we made a bind to say "Hi everyone!". Now let's make that bind again, but in blue text. The colour code for blue is ^4, so the bind has to be:
bind x "say ^4Hi everyone!"
So everything that follows the ^4 will appear in blue.

Name

When we started of playing the game, most of us started with a white name. Your name can easily be changed in the menu, when pressing escape. The only drawback is that you cannot add colours to your name this way. There are 2 other ways in setting your name: using the console and placing it in one of your cfg files e.g. your autoexec.cfg.

Console

In the second tutorial I explained how to pull down your console. In your console you can also type single commands you want to execute e.g. setting your name. Important to know is that all commands you type in console have to be preceded by a forward slash(/) and always an ENTER after entering a command! The command for setting your name is of course name. So changing your name in console is done as followed:
/name YourName

Let's say you would like to change your name to:
*NoobZor
In there we use blue (^4) and red (^1). Our command in console would have to be:
/name ^1*^4N^1oob^4Z^1or

In a cfg file

As I will explain more elaborately later on, it is useful to store all your settings (including your name) in a cfg file. Let's say you would like to add your name to your autoexec.cfg. In the second tutorial we discussed how to create it. Now we are going to add our name to it. So start by opening your autoexec.cfg.

In general if you want to store a setting in your cfg file, the command should look like this:
set setting value

So if you want to store your name in your autoexec file, it should be like this:
set name ^1*^4N^1oob^4Z^1or

I will talk about different settings, also known as cvars, later on. Another example of a setting is the sensitivity of your mouse. Let's just say you would like to set your sensitivity to the value 1. Then the following command should be in your autoexec file:
set sensitivity 1

In your autoexec the 2 examples above look like this:

If you want to try and see what your name looks in certain colours, you can try Easyrider's Nickname Generator at www.enemy-territory.com. If you have tried and found the perfect name, you can easily copy and paste the code to your cfg file.

One more important note. As you can see in any cfg file commands are not preceded by a forward slash(/). That's only when you enter commands in your console.

Variables (vstr).

Before you can actually start scripting you have to know how to use variables. In scripting we use variables to store information in, so that when we call that variable, we access the information stored in

that variable. So logically we have to store something in a variable, before we can use it. We store something in a variable by using the set command as followed:
set VariableName "command1; command2; ..."

The only thing the code above does is store the commands in that variable. This means the commands are not yet executed. You execute them by calling the variable using the vstr command, which is done as followed:
vstr VariableName

Now all the commands in that variable will be executed.

You also bind the calling of a variable to a key:
bind x "vstr VariableName"

Important note: mind the caps in your variable name. Case sensitive!

Let's do an example. In the previous tutorial we discussed how to set your name. Let's try to do that by first storing it in a variable. I'm going to call that variable myName. Here goes:
set myName "name ^1*^4N^1oob^4Z^1or"
bind x "vstr myName"

So now my name will not be changed into *NoobZor until I press x. This construction might not make any sense at this point, cause this could have also been done by 1 line of code, not using the variable construction at all:
bind x "name ^1*^4N^1oob^4Z^1or"

This does exactly the same, but the variable construction is important for when we want to make a toggle, which is nothing more than a loop. In this loop we can rotate between multiple variables.

Creating a toggle.

Like we discussed in the previous tutorial, a toggle is nothing more than a loop. This is best explained by using an example. Let's say I still use *NoobZor as my gaming name, but I have also decided to join a clan, which uses the [A] tag. So I also sometimes want to play with [A]NoobZor.

Now let's write our first real script. We want to be able to switch between these 2 names using one key. Again let's just use x in this example. We start by using 2 variables to store these 2 names in:
set name1 "name ^1*^4N^1oob^4Z^1or"
set name2 "name ^1[^4A^1]^4N^1oob^4Z^1or"

Now we have 2 situations we can switch between: variable name1 and variable name2.
What we want is when we press x the first time, it has to execute name1. The second time we press x, it should execute name2. The third time it has to execute name1 again, and so on. If we want to execute name1 the first time, you probably would think that binding x to variable name1 would do:
bind x "vstr name1"

Unfortunately this is not true, cause with this bind name2 can never be executed. So we need the the keybind to switch between the 2 variables and for that we need 1 more variable. Let's call this variable nameChanger.

We will bind x to this new variable nameChanger and want this new variable nameChanger to alternate between name1 and name2. This is done by the following construction:
bind x "vstr nameChanger"
set name1 "name ^1*^4N^1oob^4Z^1or; set nameChanger vstr name2"
set name2 "name ^1[^4A^1]^4N^1oob^4Z^1or; set nameChanger vstr name1"
set nameChanger "vstr name1"

If you look at this construction you will see that when name1 is executed, the variable nameChanger is then binded to the variable name2:

set name1 "name ^1*^4N^1oob^4Z^1or; set nameChanger vstr name2"

This means the next time x is pressed and nameChanger is executed, it will no longer execute name1, but will now execute name2. The same thing goes for variable name2. When this variable is executed, the variable nameChanger is binded to name1 again:
set name2 "name ^1[^4A^1]^4N^1oob^4Z^1or; set nameChanger vstr name1"

Now as you can see I added 1 more line of code to the script:
set nameChanger "vstr name1"

This line can be seen as an initial value. Without this line the script would not know with which variable it should start the first time we press x. With this line we tell the script, that it should execute name1 the first time we press x.

So if you look at the construction of a toggle in general, it should always look like this:
bind key "vstr mainVariable"
set variable1 "command1; command2; ...; set mainVariable vstr variable2"
set variable2 "command1; command2; ...; set mainVariable vstr variable1"
set mainVariable "vstr variable1"

Of course you can work with more variables than 2. With 3 variables it would look like this:
bind key "vstr mainVariable"
set variable1 "command1; command2; ...; set mainVariable vstr variable2"
set variable2 "command1; command2; ...; set mainVariable vstr variable2"
set variable3 "command1; command2; ...; set mainVariable vstr variable3"
set mainVariable "vstr variable1"

So you can make the toggle with any amount of variables you want.

2 important notes:
* To keep the script clear, I mostly put numbers behind my variables. Like I did in my example: name1 and name2. This is not obligatory. You can use whatever name you like for your variables.
* I used a certain sequence for the lines of code, but there is no rule for this. You can determine the sequence yourself. This would work just as well:
set variable1 "command1; command2; ...; set mainVariable vstr variable2"
set variable2 "command1; command2; ...; set mainVariable vstr variable1"
set mainVariable "vstr variable1"
bind key "vstr mainVariable"

As you can see, I now placed the line with the key bind at the end. The only important thing is that it makes sense to you! So that when you read your script later on, you still understand how it works.

One more example to make sure, it's absolutely clear.
Now we want to be able to change the sensitivity of the mouse with one key. Again let's take x. The values for the sensitivity the scripts needs to alternate between, are: 1, 1.5 and 2. The script should look like this:
bind x "vstr sens"
set sens1 "sensitivity 1; set sens vstr sens2"
set sens2 "sensitivity 1.5; set sens vstr sens3"
set sens2 "sensitivity 2; set sens vstr sens1"
set sens "vstr sens1"

Echo command.

A lot of people use the say or vsay command in combination with their scripts, which can be pretty annoying to other people playing on the server. A good example is the so-called reload script. What does it do? Well as well as reloading when they press their reload key, it also says something like: "I'm reloading!" A script like this would look like this:
bind r "+reload; say_team I'm reloading!"

This script might look cool to some of you, but people don't give a damn if you are reloading. They'd rather have you keep that to yourself. If you still think you need a confirmation to know for sure that you are reloading, use an echo command and it will not show up in the chat area, but in the echo section:
bind r "+reload; echo I'm reloading!"
By the way, this is the part where all the game info shows up, like the obituaries.
The question of course when do you really this. The example above is not something sain people would use :D. There are scripts however, where you might not be able to exactly know what happened when you executed it. A good example is the sensitivity toggle from the previous tutorial:
bind x "vstr sens"
set sens1 "sensitivity 1; set sens vstr sens2"
set sens2 "sensitivity 1.5; set sens vstr sens3"
set sens2 "sensitivity 2; set sens vstr sens1"
set sens "vstr sens1"

Of course you can notice that your mouse sensitivity changes, when you press the bided key. But if you have pressed the binded key a couple of times you might not know where you are in the script, meaning you do not know which variable will be executed next time you press your binded key. So you might have lost track of at which sensitivity you are. If you just add echo commands to this script, then you can read in the echo section what the present sensitivity value is when you press the binded key. The script above would then look like this:
bind x "vstr sens"
set sens1 "sensitivity 1; set sens vstr sens2; echo Sens: 1.0"
set sens2 "sensitivity 1.5; set sens vstr sens3; echo Sens: 1.5"
set sens2 "sensitivity 2; set sens vstr sens1; echo Sens: 2.0"
set sens "vstr sens1"

Using the binded key now would not only change your sensitivity, but would also show the selected sensitivity in the echo section, like this:

So in short: you can use the echo command to point out to yourself what's happening in your script when using it.

Key pressed down scripts (+vstr).

For scripts that require an action when a key is kept pressed down we use the +vstr command. Let me first show you how to use this command properly. Binding a key to a +vstr command should look like this:
bind key "+vstr var1 var2"

As you can see you need 2 variables, one for each situation: when the key is pressed down and when the key is not pressed down. Var1 should state what should happen when the key is pressed down and var2 should state what should happen when the key is not pressed down. So you will also have to state the actions set to these 2 variables, as talked about in the seventh tutorial. In total it should look like this:
bind key "+vstr var1 var2"
set var1 "command1, command2, ..."
set var2 "command1, command2, ..."

Let's clarify this with an example. One of the most requested scripts is the sprint while shooting script. This script automatically makes you sprint while you are shooting. A perfect example of a key pressed down script, cause you are firing your gun as long as you keep your mouse1 (left mouse button) pressed down.

You need to use 2 cvars to make this script work. The first is +attack and the second is +sprint. Of course these are the commands to enable firing your gun and sprinting. These again have 2 cvars to

disable these actions: -attack and -sprint. When you keep mouse1 pressed down, you want to enable sprinting and firing your gun and of course when you let go of mouse1, you want to disable firing your gun and sprinting. This would result in the following script:
bind mouse1 "+vstr shootOn shootOff"
set shootOn "+attack; +sprint"
set shootOff "-attack; -sprint"

In theory this script should work, but in practice you will find out that it sometimes does not. It could keep on firing when you let go of mouse1. That's why you should always use the disable commands twice, like this:
bind mouse1 "+vstr shootOn shootOff"
set shootOn "+attack; +sprint"
set shootOff "-attack; -attack; -sprint; -sprint"
This way you make sure the disable commands always work.

Note: Again it does not matter what names you choose for the 2 variables. Just use specific names that make sense to you.

Cycle script.

The cycle command is not used often, cause most people tend to prefer a toggle script over a cycle script. I'll explain why after I explain how it works.

A cycle script can quickly cycle (duh :D) through different values for 1 setting, which can also be done by a toggle script. The cycle script is a bit more compact though. In general it should look like this:
bind key "cycle setting startvalue endvalue stepsize"

Let's apply this to the same example we used for the toggle in the eighth tutorial. So we want a sensitivity cycle script with startvalue 1.0, endvalue 2.0 and stepsize 0.5. This script should be as followed:
bind x "cycle 1.0 2.0 0.5"

As you can see it a lot more compact than the equivalent toggle script in the eighth tutorial. Then why do people prefer the toggle script? Well you cannot apply any echoes to a cycle script, whereas with a toggle script you can. The advantage of an echo was discussed in the ninth tutorial.

Comment lines.

When you start writing big scripts, you want to make sure you still understand everything when you read it later on. You can add clarity to your scripts by adding comments to it. All comments should be preceded by 2 forward slashes(//). Then everything that follows it on the same line is ignored and thus not seen as a command.

In tutorial 8 we discussed toggles. At the end there was an example script with a sensitivity toggle. Now let's use that example to show you can add comment lines to it:
//Binded key
bind x "vstr sens"

//Toggle
set sens1 "sensitivity 1; set sens vstr sens2"
set sens2 "sensitivity 1.5; set sens vstr sens3"
set sens2 "sensitivity 2; set sens vstr sens1"

//Initial value
set sens "vstr sens1"

That's all there is to it.

Two commands under one key.

I often hear a lot of people complaining about having so many binds, but they still want to add more. Well you can. You'll need to know how to use the +vstr command explained in the tenth tutorial of the basic scripting tutorial section.

Let's say you have 6 basic vsay commands in your config at this moment:
bind 1 "vsay hi"
bind 2 "vsay bye"
bind 3 "vsay sorry"
bind 4 "vsay Affirmative"
bind 5 "vsay Negative"
bind 6 "vsay Oops"

These are all global chats and thus heard by everyone. Maybe you sometimes want use these for your team only. If you have a lot of scripts and binds, you might not even have another 6 keys left. You actually only need 1 extra, so we can make a key combination. Just like being able to cut text in Word with CTRL + x.

For this example I'll use CTRL as that extra key. So we want to create a script that makes it possible to use the 6 binds in combination with CTRL and make the binds teams vsays instead of vsays. My way of doing this is the easiest imo, but not the only way.

First we start of by placing the vsays above in a seperate cfg file. Let's call it vsay.cfg. We also place all the team vsays in a seperate cfg file. We will call that vsayteam.cfg. So we now have 2 files with the following content:

vsay.cfg

bind 1 "vsay hi"
bind 2 "vsay bye"
bind 3 "vsay sorry"
bind 4 "vsay Affirmative"
bind 5 "vsay Negative"
bind 6 "vsay Oops"

vsayteam.cfg

bind 1 "vsay hi"
bind 2 "vsay_team bye"
bind 3 "vsay_team sorry"
bind 4 "vsay_team Affirmative"
bind 5 "vsay_team Negative"
bind 6 "vsay_team Oops"

Now we have to make the script to activate these cfg files. We want vsay.cfg to be activated when CTRL is NOT pressed down and we want vsayteam.cfg to be activated when CTRL is pressed down. So we have to bind CTRL to a +vstr command. This little script can either be placed in your autoexec.cfg or in a seperate cfg file which you of course then exec from your autoexec file as well. This script is as followed:
bind CTRL "+vstr ctrldown ctrlup"
set ctrldown "exec vsayteam.cfg"
set ctrlup "exec vsay.cfg"

So now when you keep CTRL pressed down, the vsayteam.cfg file is executed and thus all binds in that cfg are activated. If you let go of CTRL again, vsay.cfg will be executed and thus all binds in vsay.cfg will be activated. That's all there is to it.

Class map and team autoexec files.

In the basic scripting section we discussed the autoexec.cfg file. Most mods now also support autoexec files for map, team and class. We will discuss all 3 types.
Before we start 1 important note:
All these autoexec files do not go in the etmain folder, but in the folder of the mod you play! E.g. the etpro folder.

1. Map

For all these autoexec files the spelling of the file name is essential. For the map autoexec files the file name has to be:
autoexec_mapname.cfg
So if you want to make one for oasis, then it has to be:
autoexec_oasis.cfg

Now what is it useful for. Well these autoexec files give you the possibility to have special settings for just this map. For instance some maps are darker then others. You might want to set your brightness (the so called gamma) higher for these maps. Let's say your default gamma is 3 and for radar you would like it to be 1. How do you set that up. It's done as follwed. Make an autoexec_radar.cfg and an autoexec_default.cfg. The default one is loaded when no autoexec is found for the map you are playing! Then you place the commands in those file:

autoexec_radar.cfg
set r_gamma 1

autoexec_default.cfg
set r_gamma 3

It's as easy as that!

Note:
The autoexec for maps is often used to make spawnselectors. You can put the relevant spawns in the autoexec for a specific map. CHeck the download section for a spawn selector.

2. Team

Not so hard to guess that these are autoexec_axis.cfg and autoexec_allies.cfg. This way you can set up different settings settings per team. Maybe you want to have different say and vsay binds per team. For instance:

autoexec_axis.cfg
bind x "vsay cheer ^1>^7Goooooo Axis^1!"

autoexec_allies.cfg
bind x "vsay cheer ^4>^7Goooooo Allies^4!"

3. Class

The following are class related autoexec files:
* autoexec_medic
* autoexec_engineer
* autoexec_fieldops

* autoexec_covertops
* autoexec_soldier
Using these enable you to have different scripts/binds per class. This way you could give a key a different functionality for each class.

Creating a proper name selector.

In our basic tutorials we talked about creating a toggle. This is what most people use when creating a name toggle too. This is not recommended though. You might have a toggle of 7 names. Now you want to choose the seventh name, so you start pressing your binded key to set the right name. When doing this you are changing your names 6 times, cause you want to go from name 1 to name 7. Some servers have a security setting preventing you from too many name changes in a certain amount of time(, to prevent you from lagging the server). So a script like this could get you kicked.

So in short a simple script like this will work, but is not smart:
```
bind x "vstr changeName"
set name1 "name Name_nr_1; set changeName vstr name2"
set name2 "name Name_nr_2; set changeName vstr name3"
set name3 "name Name_nr_3; set changeName vstr name4"
set name4 "name Name_nr_4; set changeName vstr name5"
set name5 "name Name_nr_5; set changeName vstr name1"
set changeName "vstr name1"
```

It would be better if you could scroll through your names until you found the one you want and then change you current name to the prefered name. So you would need 1 key to scroll through your names. Of course seeing the names in the echo section, so you know which name you have selected. And another key to actually set your new name.

Ok let's go through this step by step. First we start of by storing all the names in variables:
```
set N1 "name Name_nr_1"
set N2 "name Name_nr_2"
set N3 "name Name_nr_3"
set N4 "name Name_nr_4"
set N5 "name Name_nr_5"
```

When we execute one of these variables (i.e. vstr N1) you name will be changed to the name stored in that variable. Now we need to create a toggle to scroll through the names:
```
bind x "vstr changeName"
set name1 "echo Nick: Name_nr_1; set changeName vstr name2"
set name2 "echo Nick: Name_nr_2; set changeName vstr name3"
set name3 "echo Nick: Name_nr_3; set changeName vstr name4"
set name4 "echo Nick: Name_nr_4; set changeName vstr name5"
set name5 "echo Nick: Name_nr_5; set changeName vstr name1"
set changeName "vstr name1"
```

As you can see the only thing this toggle does at this moment is echo the names you scroll through while pressing x, nothing more! Now I want to add another key y, to actually set the name. So if I scroll through my names with x to for instance name 4 and I would press y, I want my script to set my name to Name_nr_4. In other words I want it to execute N4 when I press y. For this we need an extra variable, let's call it setName. This is the variable that will be binded to y:
```
bind y "vstr setName"
```

Now the question is how do we actually connect N1-N5 to setName. Well by adjusting the variables name1 to name5 like this:
```
set name1 "echo Nick: Name_nr_1; set changeName vstr name2; set setName vstr N1"
set name2 "echo Nick: Name_nr_2; set changeName vstr name3; set setName vstr N2"
set name3 "echo Nick: Name_nr_3; set changeName vstr name4; set setName vstr N3"
set name4 "echo Nick: Name_nr_4; set changeName vstr name5; set setName vstr N4"
```

```
set name5 "echo Nick: Name_nr_5; set changeName vstr name1; set setName vstr N5"
```

As you can see when x is pressed and name 1 is echoed, then the variable setName is connected to N1. So now when I press y (which connected to variable setName) N1 is executed. When I press x again, the second name is echoed. Now setName is connected to N2. So when I press y, N2 is executed. And so on...
So our script in total up till now is as followed:

```
bind x "vstr changeName"
bind y "vstr setName"

set N1 "name Name_nr_1"
set N2 "name Name_nr_2"
set N3 "name Name_nr_3"
set N4 "name Name_nr_4"
set N5 "name Name_nr_5"

set name1 "echo Nick: Name_nr_1; set changeName vstr name2; set setName vstr N1"
set name2 "echo Nick: Name_nr_2; set changeName vstr name3; set setName vstr N2"
set name3 "echo Nick: Name_nr_3; set changeName vstr name4; set setName vstr N3"
set name4 "echo Nick: Name_nr_4; set changeName vstr name5; set setName vstr N4"
set name5 "echo Nick: Name_nr_5; set changeName vstr name1; set setName vstr N5"

set changeName "vstr name1"
```

Now it's almost completed. It just needs one more thing. The last line of the script above is the initial value for variable changeName. As we have discussed in the toggle tutorial in our basic tutorial section, our script needs to know where to start the first time we press x. That's what this initial value is for.
If x is not pressed at least once, then the variable setName is not connected to any of the variables N1 to N5 yet. Pretty logical, cause you cannot actually set the name with y if you haven't chosen one yet with x. So we need to set the variable setName to an echo message stating, you have to choose a name first. So the initial value for setName will be:

```
set setName "echo Choose a name first using x!"
```

Now we have our total script. I also added some comment lines to make it clear which part does what. Here it is:

```
//Binded keys
bind x "vstr changeName"
bind y "vstr setName"

//Your names
set N1 "name Name_nr_1"
set N2 "name Name_nr_2"
set N3 "name Name_nr_3"
set N4 "name Name_nr_4"
set N5 "name Name_nr_5"

//Toggle
set name1 "echo Nick: Name_nr_1; set changeName vstr name2; set setName vstr N1"
set name2 "echo Nick: Name_nr_2; set changeName vstr name3; set setName vstr N2"
set name3 "echo Nick: Name_nr_3; set changeName vstr name4; set setName vstr N3"
set name4 "echo Nick: Name_nr_4; set changeName vstr name5; set setName vstr N4"
set name5 "echo Nick: Name_nr_5; set changeName vstr name1; set setName vstr N5"
```

//Initial values
set changeName "vstr name1"
set setName "echo Choose a name first using x!"

Creating a class selector.

I'll be discussing the newer style, cause it's far simpler than the older style.

First the basic commands:

1. Selecting a team

A team can be selected with the team command. There are 3 possible teams, with the corresponding commands:
* spectator: team s
* axis: team r
* allies: team b

So if you wanted to make a bind to quickly go spectator e.g. for when your phone rings using your F12 key, it would be like this:
bind F12 "team s"

Or if you enter a server and you want to quickly join axis using F11:
bind F11 "team r"

One important note:
The bind above does not contain a class and weapon choice and thus makes you a soldier with SMG by default.

2. Selecting a class and weapon

A class and weapon can be selected with the class command, which is as followed:
class cls wpn

As you can see the class command is followed by cls, a letter to choose the class and wpn, a number to choose the weapon.
For cls the following options are possible:
* m: medic
* e: engineer
* f: field ops
* c: covert ops
* s: soldier

The number wpn depends on the class you are using. It also depends on the mod you are playing, cause some mods like NoQuarter have additional weapons like the Winchester and Venom. I'll state all the possibilities per class:

Medic

1: SMG

Engineer

1: SMG
2: Rifle nade
Additional (for mods like NQ):
3: Shotgun (Winchester)

Field ops

1: SMG
Additional (for mods like NQ):
2: Shotgun (Winchester)

Covert ops

1: Sten
2: FG42
3: K43 or M1 Garand
Additional (for mods like NQ):
2: Bar(allies)

Soldier

1: SMG
2: MG42
3: Flamethrower
4: Panzerfaust
5: Mortar
Additional (for mods like NQ):
1: STG44(axis) or Bar(allies)
2: Browning(allies)
4: Bazooka(allies)
5: Granatwerfer(axis)
6: Venom

Now let's say you play medic almost all the time and you want to make 2 binds. One for going medic on the axis team with F11 and one for going medic on the allied team with F12. Using the commands above, that would look like this:
bind F11 "team r; class m 1"
bind F12 "team b; class m 1"

Now let's do an easy example of combining the team and class command with a toggle to make a simple class selector.

Let's say you have 3 favorite classes you always play:
* Medic
* Engy with SMG
* Covert ops with sniper rifle

First you will need a toggle to choose your team, which we will do with F11:
bind F11 "vstr teamSelect"

```
set axis "set teamSet team r; set teamSelect vstr allies; echo ^0>^7Axis Team Set^0!"
set allies "set teamSet team b; set teamSelect vstr axis; echo ^0>^7Allies Team Set^0!"
set teamSelect "vstr axis"
```

As you can see, the team is set to a variable called teamSet instead of just executed. And to make clear what I mean by just executed, I did not use the following:

```
set axis "team r; set teamSelect vstr allies; echo ^0>^7Axis Team Set^0!"
set allies "team b; set teamSelect vstr axis; echo ^0>^7Allies Team Set^0!"
```

This is because I don't want the team to be set until the class and weapon are also chosen. If I would execute the command team right away and I would be to slow chosing the class and weapon before respawn time is over, I would spawn default as a soldier with SMG. That's why I did not execute it, but set it to a variable teamSet. And the team will be executed by executing the variable teamSet at the same time when the class command is executed.

Now we'll make a toggle to choose the class. This will be done with F12:

```
bind F12 "vstr classSelect"
set class1 "vstr teamSet; class m 1; set classSelect vstr class2; echo ^0>^7Going Medic^0."
set class2 "vstr teamSet; class e 1; set classSelect vstr class2; echo ^0>^7Going Engineer^0."
set class3 "vstr teamSet; class c 3; set classSelect vstr class2; echo ^0>^7Going Covert Ops^0."
set classSelect "vstr class1"
```

As you can see, the team is set with vstr teamSet right before the class command is executed. This way team and class are set.
You could say it's finished now, but there is just one line missing. The initial value for the variables teamSelect and classSelect are set, but not for the variable teamSet. If someone tries to choose a class before he/she has chosen a team, it won't work. By setting an echo message to the teamSet variable as an initial value, the user can be warned that he is trying to choose a class before he/she has chosen a team. This should be as followed:

```
set teamSet "echo ^0>^7You have not chosen a team yet^0!"
```

We now have the complete script. Here it is in total with comment lines added:

```
//Binded keys
bind F11 "vstr teamSelect"
bind F12 "vstr classSelect"

//Team select
set axis "set teamSet team r; set teamSelect vstr allies; echo ^0>^7Axis Team Set^0!"
set allies "set teamSet team b; set teamSelect vstr axis; echo ^0>^7Allies Team Set^0!"
set teamSelect "vstr axis"

//Class select
set class1 "vstr teamSet; class m 1; set classSelect vstr class2; echo ^0>^7Going Medic^0."
set class2 "vstr teamSet; class e 1; set classSelect vstr class2; echo ^0>^7Going Engineer^0."
set class3 "vstr teamSet; class c 3; set classSelect vstr class2; echo ^0>^7Going Covert Ops^0."
set classSelect "vstr class1"

//Error message
set teamSet "echo ^0>^7You have not chosen a team yet^0!"
```

As you can see the commands team and class are fairly easy to use. You can make the class selector as extensive (and complicated) as you want. In the download section you can find the Class selector new style. Studying this class selector, will give you even more insight in creating a proper class selector. This file uses every class and weapon available.

Confirmation echoes.

When executing a script or your autoexec, it's nice to know if it has actually been executed. You can use the echo command for this purpose. So you want to echo some information, when your autoexec is executed. Something like:
>Autoexec loaded!

The only thing you need to do is add the echo line to your autoexec:

Now as you can see it's not binded to a key or set to a variable. It's just a command, which will be executed directly when your autoexec is executed. So if this line is echoed, you know your autoexec is loaded. It will look something like this in the echo section:

You can even go a bit further with this. Let's say you made a config with an autoexec in which 3 other cfg files are executed: settings.cfg, vsays.cfg and scripts.cfg. If you place a confirmation echo is all 3 cfg files, then all 3 will be automatically echoed, cause the autoexec executes all 3 cfg files. The setup could look like this:

Autoexec.cfg:
As you can see a confirmation echo and execution of the 3 cfg files:

And a confirmation echo in each cfg file:
settings.cfg

vsays.cfg

scripts.cfg

Resulting in the following echo, when executing he autoexec.cfg:

Source yvo