```
________//|
|  THE  ||\
\________==
== GLITCH
CODEC
TUTORIAL ==
==-------\\
 BY NICK
BRIZ //----
---
```

```
_________//||  The  ||\\_________
==== GLITCH CODEC TUTORIAL ====
-------\\ by Nick Briz //-------
```

(ɔ)2010 - 2011
copy<it>right

```
<!---------------------------//++\\----------------------------!>
```

a tutorial/essay on the technical, theoretical, and critical process of glitch art

```
<¡---------------------------\\++//----------------------------¡>
```

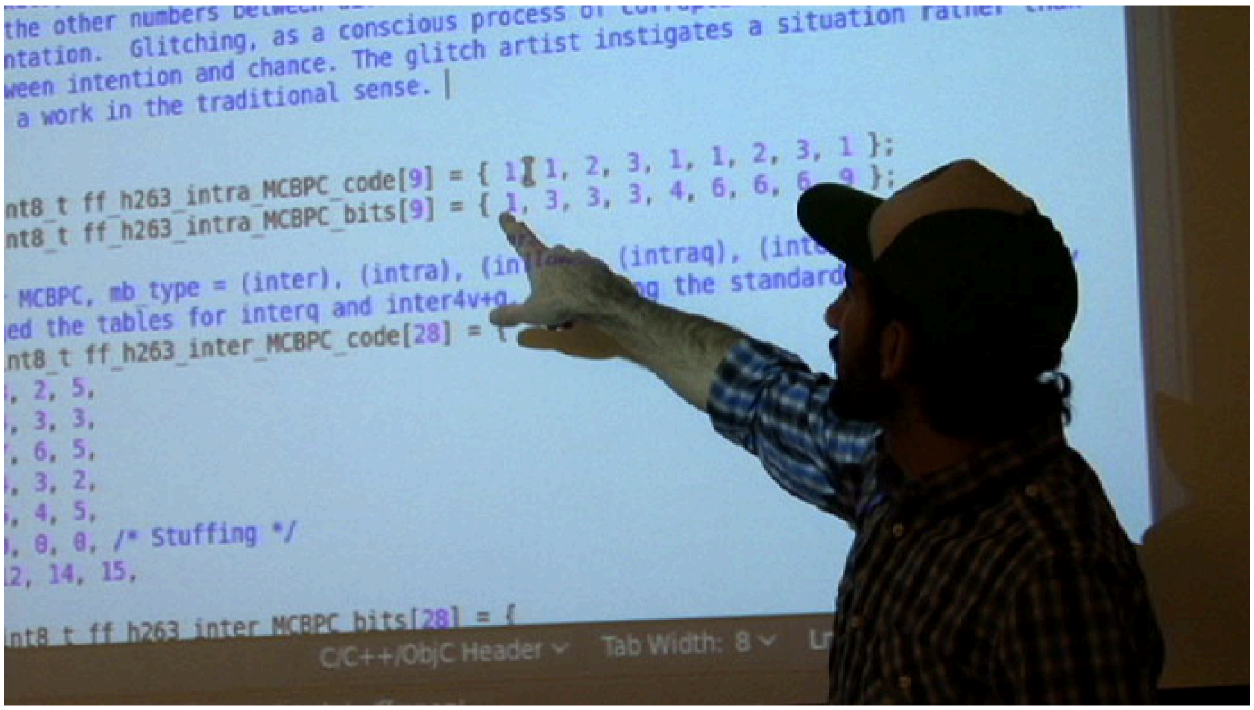
```
/******************************************************************\
 *                                                                *
 *                                                                *
 *      for downloads + more info visit: nickbriz.com/glitchcodectutorial      *
 *                                                                *
 *                                                                *
\******************************************************************/
```

```
<!DOCTYPE tutorial>

//
```

This tutorial is equal parts workshop and artist lecture—the subject of which is "glitch art." Glitch art is the aestheticization, recognition, and/or instigation of a glitch—a break and/or disruption of the expected flow of a system. Glitching, as a process, is inherently reliant upon technology yet surprisingly accessible and executable. Glitch lends itself to pedagogy as much as it serves as a ruse to traditional modes of artistic instruction that codify random acts of creativity. Glitch art, both formally and conceptually, is resistant, foregrounding a critical relationship to the digital culture in which we find ourselves mired.

This workshop/lecture is titled the Glitch Codec Tutorial. Here I demonstrate how to create the "glitch codec"—a hacked piece of software I use to make intentional glitches. The Glitch Codec Tutorial is one way to experience glitch art. The Glitch Codec Tutorial can be used to make glitch art, but it is not a tool in and of itself. Rather, it is a means to a tool—or, more appropriately, a means to a method[ology] of production. I say method[ology] because as a tutorial it is principally about laying out a set of easy to follow technical instructions for creating your own Glitch Codec. This tutorial, however, is more than just a "how to"—it is a full disclosure of my personal process. This means that I share not only my tools, techniques, and tricks but my feelings, philosophy, and ethic as well. Issues ranging from the open-source and copy-left movements to the phenomenological impact of codecs on our moving image culture are addressed while accessibly guiding participants through the hacking/data-bending process. It is equal parts technical lesson and theoretical lecture.

```
<!--- // --->
```



```
the other numbers betwee
ntation.  Glitching, as a conscious process of
ween intention and chance. The glitch artist instigates a situation rather
 a work in the traditional sense. |

nt8_t ff_h263_intra_MCBPC_code[9] = { 1 1, 2, 3, 1, 1, 2, 3, 1 };
nt8_t ff_h263_intra_MCBPC_bits[9] = { 1, 3, 3, 3, 4, 6, 6, 6, 9 };

 MCBPC, mb_type = (inter), (intra), (in      (intraq), (inte
ed the tables for interq and inter4v+q      g the standard
nt8_t ff_h263_inter_MCBPC_code[28] = {
, 2, 5,
, 3, 3,
, 6, 5,
, 3, 2,
, 4, 5,
, 0, 0, /* Stuffing */
2, 14, 15,

nt8_t ff_h263_inter_MCBPC_bits[28] = {
```

```
/*****************\
 *               *
 *  Introduction  *
 *               *
\*****************/
```

```
// Glitch Art
```

We should start by defining glitch art more accurately. In order to do this you'll need an image file and a text editor. If you don't have any image files on your computer download one from the Internet (your standard jpeg will do). You could do a google image search for some image. Right mouse click the image, or ctrl click in Mac, and save the image to your desktop. Make a back-up of the image by copying and pasting the file.

Double click that image file. Your computer "knows" that this file is an image file and so it assumes you would like to view this file with your default application for displaying images. A jpeg, like any digital file, is really just a container for 1's and 0's (binary code, digital data) and so could be interpreted by other software which doesn't necessarily have to be an application for displaying images.

Now open your image with TextEdit(Mac) or Notepad/Wordpad(PC) either by opening TextEdit then clicking on File > Open, and navigating to your jpeg file. Or by right mouse clicking (or ctrl clicking) on the jpeg image and selecting Open With > Other... and choosing TextEdit. You should see a garbled mess of text. This text is an ASCII representation of the data contained in your jpeg file, it's TextEdit's interpretation of your file. Scroll down to the middle of the file and type some text of your own. Then save your file (maintaining the .jpeg or other image extension, do not save it as a .txt or .rtf)

Reopen your image file in your default application for displaying images after having saved the changes you made to it with TextEdit and you should now be looking at a corrupted image. You can make additional changes with TextEdit. Try copying a large chunk of text and pasting it a couple of times elsewhere in the file, save it, and reopen your image. You could also use the "find" and "replace" function in your text editor to swap one character (the letter "a" for example) with another character (the letter "z"), save that, and reopen your image. This will effect an even more drastic change.

This is an easy and relatively quick method for intentionally glitching files (specifically images). The Glitch Codec Tutorial, however, is a bit more complex and time consuming but much more versatile and potentially much more rewarding. By complex I don't mean difficult, but simply that it is more involved. In this tutorial I will show you how to create a customized/hacked codec through which to process video files in addition to discussing how this is possible and perhaps why it should be done. This will result in unpredictable yet controllable glitches.

// Theoretical Framework

Glitch Art is often aestheticized and fetishized technological [human] errors or anticipated accidents that can produce unintended [desired] results.

J. Hillis Miller, a prominent American deconstructionist, has described deconstruction this way: **"Deconstruction is not a dismantling of the structure of a text, but a demonstration that it has already dismantled itself. Its apparently-solid ground is no rock, but thin air."**

In *some* ways this is the kind of approach to glitch that I'm advocating for in this tutorial. Glitching is a tactful exploitation of the systems at play in the digital sphere. This Glitch Codec Tutorial exploits a specific hack. Hacking is about making use of other uses; glitching is about hacking—so glitch art can be about raising awareness of these "other" uses by deconstructing digital conventions. This awareness establishes a critical relationship between the user and the machine—one which this tutorial hopes to present for others by creating a process which, though systematic, destroys systems, or rather instigates self-destruction. This hack always exists in the present and never in the past because every use of the "glitch codec" necessitates a new destruction of a codec [file] at the code level - creation by destruction. *Destruction* in this case is used to refer to this "demonstration" of the system's appeared self-dismantilization. The reason I say "appeared" self-dismantilization is because a computer can not make mistakes, the user inputs and the machine outputs, the glitch is the unexpected output which, by catching us off-guard, makes us aware of the medium, its structure, and its politics. The error, therefore, is a human one, which can belong to either the programmer or the user. But, we prefer to place the onus on the machine and refer to this occurrence as a "glitch." When one intentionally and desirably instigates the glitch what results can be understood as glitch art. Given the nature of glitches the means, method, and situation in which it's invoked can establish critical relationships to digital media and culture, creating a potentially democratic space for aesthetic and conceptual dialogue. But if the digital/information revolution is to bring with it democratic possibilities certain freedoms are essential.

The Glitch Codec Tutorial itself can be discussed as glitch art but it is important to note that this tutorial is something that is always experienced not just observed - it is a process not a product.

<!--- // --->

```
/*****************\
 *               *
 *     Step 1    *
 *               *
\*****************/
```

```
// AN OPERATING SYSTEM ON DVD
```

In order to take part in this tutorial you need the Glitch Codec Tutorial DVD. You can download the .iso file on my site (http://nickbriz.com/files/GlitchCodecTutorial.iso) to burn your own copy of the DVD.

After downloading the .iso you can burn the disk image to DVD by either using disk utility on a mac or your default DVD burning software on a PC. The Ubuntu forums have easy step by step instructions for burning ISO files to disk on various operating systems (https://help.ubuntu.com/community/BurningIsoHowto) The Ubuntu forums are a great place to go to with GNU/Linux questions, it was a huge resource for me while creating this project.

```
// While that burns: {Open Source || Commercial/Proprietary Software}
```

What you are downloading is my customized version of the Ubuntu OS a popular user-friendly GNU/Linux distribution. It's important to note the particular environment chosen for creating the glitch codec. As opposed to more popular operating systems like Mac OSX and Microsoft Windows, Ubuntu is an operating system based on the GNU/Linux distribution. There are many similarities and differences between these operating systems (or OS) but the most significant one is that Ubuntu (like other GNU/Linux distros) is open source while Mac OSX and Microsoft Windows are commercial/proprietary software. Most commonly people discuss open source as being "free" and proprietary software as having to be paid for, but the "free" is more than just a financial issue. It refers to freedoms one has when using the software. It’s important to note that when you buy proprietary software you actually aren't "buying" the software the way you would buy a car, rather you are "licensing" that software which is more like leasing a car—you can drive it but you can't re-sell or make any significant changes to it. This is an issue for us because with this particular project we need to be able to make changes to software at the source.

This is where the difference between source code and object code becomes important. When computer programmers write software they write it in "source code" which like all human language can be learned, read, and written by humans. Later that source code is compiled by a "compiler" and turned into "object code" which looks like gibberish to humans but can be read and executed by a computer. When you invoke a piece of software the code beneath the interface being executed by the computer is the object code. When a programmer writes or makes changes to an existing piece of software (s)he is working with the source code. In addition to very tight licensing rules and restrictions, proprietary software only gives you access to "binaries" or object code which is what you are running on your computer when you use something like Microsoft Office. While you've been licensed to use this software on your computer you can't make any changes to it to suite your specific needs (as you would when you paint your car). You are not granted access because it is not yours, you've only been given permission to use it—a specific and confined relationship with the software, as dictated by the manufacturer's special interests, is imposed on you. With open source in addition to having free access to object code you are given access to the source code of any given piece of software granting you and others the freedom to use and modify the software as you see fit. This is the case with Open Office for example which is a free and open-source alternative to Microsoft Office.

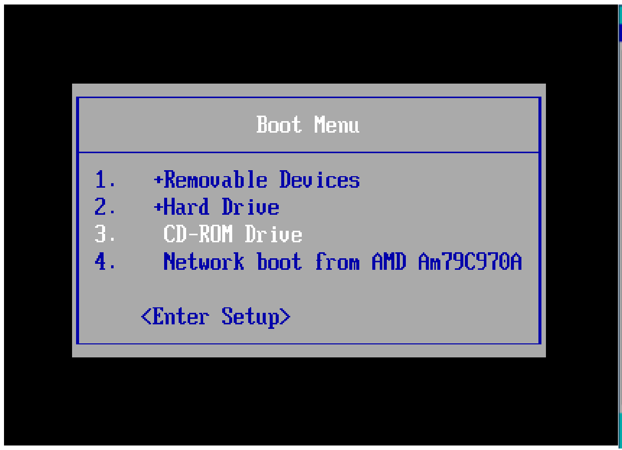
`{ Free as in free speech not as in free beer (Richard Stallman)}`

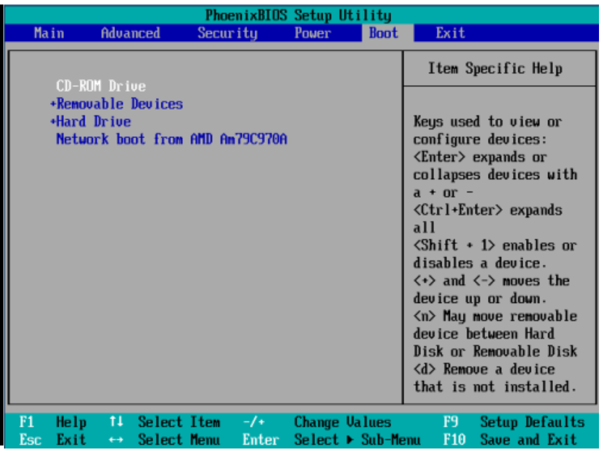
`// Booting from the Glitch Codec Tutorial DVD`

Once the DVD has finished burning, restart your computer with the DVD still inside in order to boot the OS from the disk. What this means is that your machine will be running the Operating System on the DVD rather than running its normal OS. This won't make any permanent changes to your computer.

`<Reboot>`

`<PC>` Press F12 (or other key if indicated during start up) to enter the BIOS boot menu, then select the option to boot from your DVD/CD-ROM drive `</PC>`



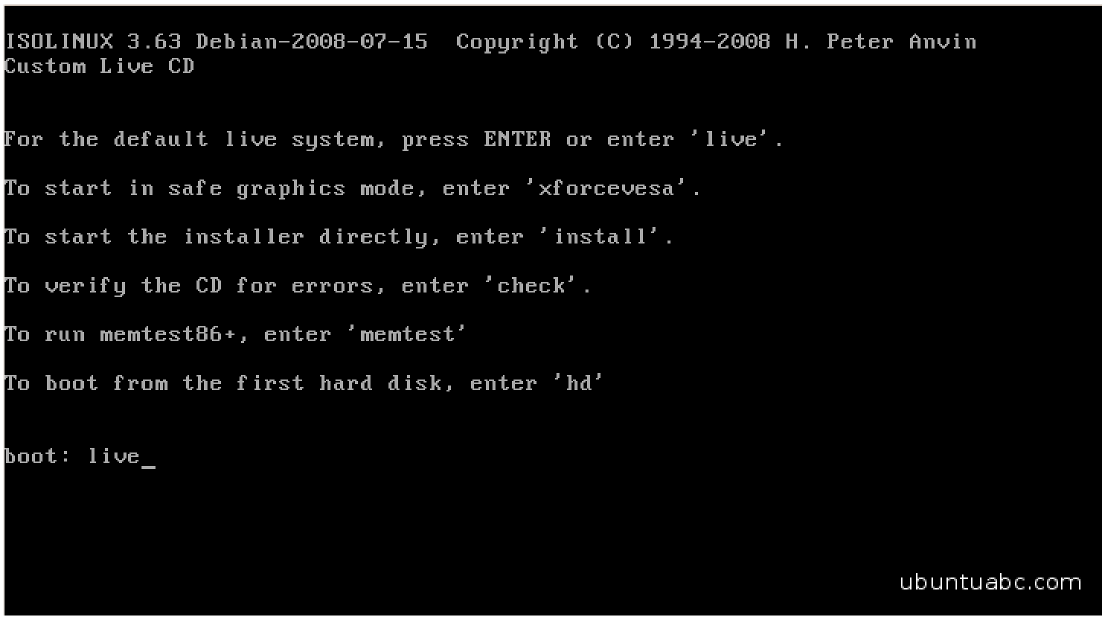

PhoenixBIOS Setup Utility
Main Advanced Security Power Boot Exit
CD-ROM Drive
+Removable Devices
+Hard Drive
Network boot from AMD Am79C970A
Item Specific Help
F1 Help ↑↓ Select Item -/+ Change Values F9 Setup Defaults
Esc Exit ↔ Select Menu Enter Select ▸ Sub-Menu F10 Save and Exit

Keys used to view or configure devices:
<Enter> expands or collapses devices with a + or -
<Ctrl+Enter> expands all
<Shift + 1> enables or disables a device.
<+> and <-> moves the device up or down.
<n> May move removable device between Hard Disk or Removable Disk
<d> Remove a device that is not installed.

`/* a couple examples of what a your PC BIOS should looks like */`

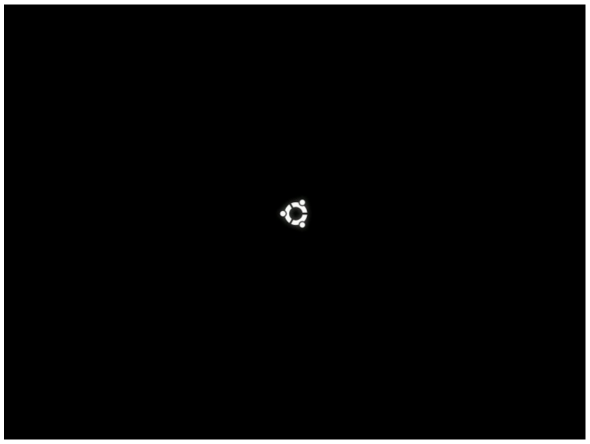
`<MAC>` Hold down the C key immediately after reboot. Hold it down until the black screen below appears. If it does not appear restart and try again. `</MAC>`

You will have a few choices regarding how to run the CD: live, xforcevesa, install, check, memtest, and hd - choose the first option by typing 'live' and pressing enter, or simply by pressing enter:



```
ISOLINUX 3.63 Debian-2008-07-15  Copyright (C) 1994-2008 H. Peter Anvin
Custom Live CD


For the default live system, press ENTER or enter 'live'.

To start in safe graphics mode, enter 'xforcevesa'.

To start the installer directly, enter 'install'.

To verify the CD for errors, enter 'check'.

To run memtest86+, enter 'memtest'

To boot from the first hard disk, enter 'hd'


boot: live_
```

You’ll then see the Ubuntu logo below. It should pulsate for a while (sometimes a long while) and then boot into the OS.



When it loads up you'll be prompted for a password. type in: 010 .

> **NOTE:** if you decide you want to stop the tutorial at any moment and return to your regular desktop simply shutdown or restart the computer by clicking the on/off logo on the top right-hand corner of the screen next to the user-name. After reboot your disk will eject and you will be prompted to hit "enter" to finish the reboot - if you are not prompted and are stuck on a black screen after your disk has ejected simply hit enter and wait a minute - your system will finish restarting/or shutting down.

```
/*****************\
 *               *
 *     Step 2    *
 *               *
\*****************/
```

// CODECS

For various reasons (web-streaming, DVD encoding, Television broadcasting, etc.) video files need to be compressed, that is that they need to look and play as best as possible but need to be as small a file as possible in order to efficiently achieve said reasons. Codecs (coder/decoder) are a series of compression algorithms used to achieve a compromise between a file's size and quality based on assumptions made about what we won't notice or don't need to see/hear in a given media file. FFmpeg is a free and open source software tool used to convert a video from one file format to another, this is done by compressing videos with different codecs. Currently we have two versions of ffmpeg running—one simply called "ffmpeg" and another called "glitchcodec" which at the moment is exactly the same as ffmpeg (a copy). We will be hacking the codecs used by ffmpeg to create our custom "glitchcodec".

Before we learn how to break things we need to know how to make them.

Lets run a video through the “glitchcodec” (which right now is just a copy of ffmpeg). We’ll need a sample video. I’ve got some in the Home Folder you can use. Click on “Places” on the top left hand corner of your screen. Then click on “Home Folder.” If you have your own videos you can drag and drop them from a USB stick into the “Home Folder.” You could also download videos from the Internet. I’ve included a few FireFox plug-ins which let you easily download videos from YouTube. If you choose this option, don’t forget to save the videos you download to the Home Folder. Also, make sure to give it a small name with no spaces (ex: testvid.mp4).

> **NOTE:** To clarify, anything you "save" during this tutorial is only being saved temporarily in your RAM. As I mentioned before you're running this OS live from a DVD—no changes are being made to your hard-disk. This means files you "save" aren't going to be there after you restart your computer. For this reason I recommend using a USB Stick or external hard-drive (formatted FAT32) to save anything you might want to keep after this session.

// THE TERMINAL

Once you’ve chosen a sample video, open up a terminal by clicking the icon on the desktop or by going to the menu on the top left again and select Applications > Accessories > Terminal. The Terminal is a command line application, it’s a way of interfacing with the computer that doesn’t pretend to be an object from real life (like a folder, trash-can, etc.).

The Terminal will be our tool for both running our "glitchcodec" as well as compiling(making) it. You should see a blank window with glitch@glitch:~$

Type glitchcodec and hit enter. Some information pops up letting us know that the glitchcodec is in-fact installed on our system, and that it’s really just a copy of FFmpeg.

Now lets run our test video through it. type:

```
glitchcodec -i Aladdin.mov newvid.mov
```

What we're doing here is telling the computer to run "glitchcodec" then to “input” (-i) our test video, which in this case is Aladdin.mov but you can use any other video you place in the home folder. Then by typing in “newvid.mov” we’re saying that after compressing our input video (Aladdin.mov) and running it through our glitchcodec, name the output “newvid.mov” You can give the output any name, so long as you put “.mov” after it.

It's important to note that the Terminal here is case sensitive, make sure to match exactly the lower/upper case text of the files you are importing—including the case of the extension. This means that if your video is "Aladdin.mov" and in the Terminal you write "aladdin.mov" it won't work because the “a” needs to be “A”

If you typed that all in correctly in your Terminal and hit enter, you’ll see a bunch of text fill your Terminal and then stop. Open up your Home Folder (Places > Home Folder); there you’ll find your new video (newvid.mov). Click it and watch it. The new video should look more or less the same as the original only a bit more compressed which normally introduces some slight artifacts.

The reason nothing looks glitched yet is because we haven’t hacked any codecs. This is our next step.

```
/*****************\
 *               *
 *     Step 3    *
 *               *
\*****************/
```

// HACKING CODECS

Perhaps the most important part of the Glictch Codec process is hacking the codecs. You'll find these in the following directory Places > Home Folder > ffmpeg > libavcodec

In the libavcodec folder you'll find all the source code for the codecs that ffmpeg (and by extension the glitchcodec) can use to compress (glitch) video. I should point out at this point that, depending on what country you're doing this in, this may be illegal. For example if you're partaking in the tutorial in the United States you'll be breaking some intellectual property laws. However if you're in India, the Philippines, South Africa, and New Zealand you are not. I should clarify what might seem to be an ethical issue: hacking these codecs to make glitch art is NOT wrong, it simply may be illegal. Not only is it not wrong, but (in part because it is illegal) it's an absolutely critical gesture one should make when examining their relationship to digital technologies. These kinds of critical gestures are not uncommon within a glitch practice. Because the United States economy largely depends on intellectual property it has stricter (arguably absurd) copyright laws, and so what is perfectly legal and seen as essential for progress in some countries, is considered illegal in others.

Now that you've been warned, search through the directory of codecs and open a file named "h263data.h" We'll start by working with this particular codec (h263). If you scroll down in that file just a bit you'll noticed I left some "comments" to guide you along the way. Just below the comment you'll see some code which reads:

```
const uint8_t fff_h263_intra_MCBPC_code[9] = { 1, 1, 2, 3, 1, 1, 2, 3, 1 }
```

As suggested this is a good place to start experimenting. Try changing that first "1" into a "2" or "9" or any another single digit number (only change the pink numbers)Then hit save.

> **NOTE:** Like we did before while data-bending image files, it's important to make a back-up of any file you'll be experimenting on. I've already made a back-up of this particular file, which you'll find in the Home Folder. It's named "BACKUP-h263data.h", so if you're hacking gets out of hand, you can always replace it with this file.

After you've made and saved the change open up another Terminal. As I mentioned before the Terminal is where we'll be running our glitchcodec and also compiling it. Because we've made a change to the source code we now need to "re-compile" the glitchcodec in order for the changes to take effect. Start by typing the following into your new Terminal:

```
cd ffmpeg
```

"cd" stands for "change directory." This command will redirect the Terminal into the ffmpeg folder. Now type:

```
make install
```

This will begin the re-compiling process for our "glitchcodec." Like before, you'll start to see text flowing through the Terminal. This may take a little while.

As I mentioned before codecs work by making assumptions about what we "won't notice" or "don't need to see." In some cases it scans a video frame by frame and averages out blocks of pixels, this is called "spacial compression." In other cases codecs might look at the differences between frames, and remove pixels that don't change much, this is known as "temporal compression."

These assumptions/compression algorithms, "deeply affect the life of images and media today", says Adrian Mackenzie a professor at Lancaster University who's done a great deal of research and writing on the subject of codecs. He explains that, "at a phenomenological level, they [codecs] deeply influence the very texture, flow, and materiality of sounds and images (...) Codecs catalyze new relations between people, things, spaces, and times in events and forms." The role codecs play in our ways of seeing the world is more pervasive than one would imagine. Nearly every bit of media content we encounter (television, DVDs, CDs at a store, mp3's on our Ipods, videos on the internet, etc) have been compressed with codecs. Its importance is made clear from an economic standpoint, Mackenzie points out that MPEG-2, a widely used codec (commercially and otherwise), "is a mosaic of intellectual property claims 640 patents held by entertainment, telecommunications, government, academic, and military owners according to Wikipedia. The large patent pool attests to the economic significance of MPEG-2 codecs."

When our new glitchcodec has finished compiling open up a new Terminal window and type:

```
glitchcodec -i Aladdin.mov glitchAl.mov
```

Which is telling the computer to run our Aladdin video through our newly modified "glitchcodec" and output a video called "glitchAl.mov" Our new video should now playback glitched! If for whatever reason it doesn't, go back to the h263data.h file and make some more significant changes, and try again.

The kind of glitch which resulted from our hack is specific to the codec that we modified (h263) and the specific section within this specific file (h263data.h) we changed. Making changes in other sections of the h263data.h file and making changes in other files within the libavcodec folder will yield different results. The possibilities here are seemingly endless (I've been working on the Glitch Codec for two years now, and still haven't exhausted all it's potential). It's also important to note that what we are seeing and calling a "glitch" is the corrupted, or in this case misencoded, file (glitchAl.mov) in combination with the particular media player we are viewing it through. If you take this same file (glitchAl.mov) save it on a memory stick and open it on another computer through various media players you'll notice that you often get very different results—some players will refuse to open the file and our "glitched" video will be replaced by an "error message."

```
/*****************\
 *               *
 *     Step 4    *
 *               *
\*****************/
```

// MY WORKFLOW

In this last step I'd like to share my work flow as an example of how I imagine the "glitch codec" process might work. We start by opening a few windows: two Terminals, one text file, and two folders. The first folder is Places > Home Folder, this will be where we'll be importing videos from and outputting videos to. The second folder is Places > Home Folder > ffmpeg > libavcodec, this is where we'll be opening the files we'll be hacking. In there we'll find the h263data.h file, this is the text file we'll be opening and continuously modifying in this tutorial. Then we'll open up a Terminal, Applications > Accessories > Terminal, in this first Terminal we'll be re-compiling the glitchcodec after every modification we make to the h263data.h text file. So we'll start by typing:

```
cd ffmpeg
```

and then pressing enter, which as explained before redirects us into the ffmpeg folder where our "make" file resides. Then we'll open up a second Terminal which we'll be using to run our glitchcodec.

After all these windows are opened our work-flow will look as follows:

< 1 > make a change to the h263data.h file, remember only to change the pink numbers. If you scroll a bit more through the file you'll notice I've made some other comments. After making your change hit save.

> **NOTE:** you can also make changes to other codec files, however if you do you have to specify in the Terminal that you'd like to use this codec instead of the default h.263, to do this you have to add "-vcodec" and the name of your codec, for ex: [`glitchcodec -i Aladdin.mov -vcodec h263p glitchAl.mov`] (don't forget to always have an unmodified back-up of any file you intend to hack).

< 2 > apply the change to the glitchcodec by recompiling it—to do this go to the Terminal which has been "cd" to "ffmpeg" and type "make install", wait a few seconds for your new glitchcodec to compile.

< 3 > when this is done switch over to the other Terminal which you are using to compress video through the glitchcodec and type into the command line as explained before: [`glitchcodec -i Aladdin.mov glitchAl2.mov`] (Remember to designate "-vcodec" when necessaryy)

< 4 > Switch over to the Home Folder window and play your newly exported video (glitchAl2.mov).

< 5 > if you like what you see back up this file on a USB stick or external hard-drive.

< 6 > repeat the steps, this time trying another hack in step < 1 >

It's important to note what I said before about the glitch being a combination of the misencoded file and the player we're viewing the video file in. You might view your newly outputted file and like what you see, then save that file on a USB stick only to later realize that it looks/behaves differently on another machine or player or that it doesn't play at all. For this reason I've included a screen-recording software on the desktop called gtk-recordMyDesktop. This application lets you record whatever happens on your screen. You can select the specific region you want to record (i.e. the media player window) and record it as it plays. This will give you a stable recording of the glitch that occurred. This is not the glitch itself but rather a documentation of the glitch, and so it will play exactly the same way every time you play it. For more on this piece of software check out their website: http://recordmydesktop.sourceforge.net/about.php.

Now that you're creating output with the glitchcodec there's one more important point I should explain about the nature of digital files, which is really similar to the nature of ideas. There's a great Thomas Jefferson letter which gets quoted a lot in reference to the nature of ideas:

> *"If nature has made any one thing less susceptible than all others of exclusive property, it is the action of the thinking power called an idea, which an individual may exclusively possess as long as he keeps it to himself; but the moment it is divulged, it forces itself into the possession of every one, and the receiver cannot dispossess himself of it. Its peculiar character, too, is that no one possesses the less, because every other possesses the whole of it. He who receives an idea from me, receives instruction himself without lessening mine; as he who lights his taper at mine, receive light without darkening me. That ideas should freely spread from one to another over the globe, for the moral and mutual instruction of man, and improvement of his condition, seems to have been peculiarly and benevolently designed by nature, when she made them, like fire, expansible over all space, without lessening their density at any point and like the air in which we breath, move and have our physical being, incapable of confinement or exclusive appropriation."*

This is also true about digital files. In economics when a good is limited or scarce by nature it's known as a rivalrous good. This means first that my use or consumption of a good prevents others from using that good, and second that in order to produce an additional unit of that good we need to use some limited social resource which we could otherwise use for something else. For example, a chair. While I'm sitting on a chair no one else can sit on that chair, and if we want to make another chair we would have to use resources like wood and nails which we would otherwise use to make something else (like a table). The cost of producing that additional unit is known as it's "marginal cost." A non-rivalrous good is something with a marginal cost of zero, for example ideas or information as Jefferson points out. Digital files are also non-rivalrous. We can infinitely copy digital files at no cost, additionally my use or consumption doesn't prevent anyone else from using or consuming that digital file.

Economically we understand that when we place the price of a non-rivalrous good above it's marginal cost of zero we create an inefficient market. This is exactly what the proprietary/ commercial software model does. It attempts to impose an unnatural scarcity on digital software. Once we know this we can understand piracy to mean a refusal to comply with the industry's forced scarcity agenda. This agenda doesn't just stop with software. It's also imposed on video files, music files, and any other kind of digital file via licencing and copyright. This is why, in recent years, we've seen a kind of back-lash towards copyright known as the "copy-left" and what I like to call the "copy-middle." The copy-middle (often referred to as the copy-left) are those who propose alternatives to the current copyright system. So for example, Creative Commons which works within copyright law, while allowing for certain creative freedoms, like appropriation and remixing for non-commercial uses. The copy-left (often reffered to as the copy-far-left) are those who are opposed to copyright all together, they disagree with the notion that ideas could be a form of exclusive property at all.

Because of the nature of this project, I'm releasing this tutorial copyleft, or perhaps more appropriately Copy<It>Right which is a copyleft license/ethic established in the 70's in Chicago by video artist Phil Morton and Dan Sandin (from whom this project takes a lot of influence and inspiration). For that reason I'm going to ask that anything you do that comes out of the use of this tutorial—any interesting hacks you find, or videos you make—that you share that (email it to me!), and release it under Copy<It>Right. Of course, given the nature of the copy-left ideology, you don't have to do that.

<!--- // --->

```
/*****************\
 *               *
 *   Conclusion  *
 *               *
\*****************/
```

The naturally occurring glitch is an unexpected and often perplexing event. We've come to understand that the intentional instigation of this kind of occurrence can be called glitch "art". Glitch Art often establishes a healthy critical relationship towards digital technologies; one granted and encouraged by the freedoms emphasized within certain spheres (the open source community for example). This freedom, which is essential to said critical relationship, is not only withheld but hidden in other spheres (the commercial and proprietary software markets). Glitching can be a means to protest against these entities and their special interests which deny us these freedoms. Glitching is a critique of the medium's structure (code), material (interface), and politics (open/closed, free/non-free, owned/commons, etc). The critical and intentional act of glitching demands this freedom in order to instigate, to explore, and to relate.

As mentioned before, the glitches which occur by means of this tutorial exist as a result of two things: first, the misencoded file outputed from our hacked glitchcodec and second, the player (software and/or device) used to view this file. In this way the glitch is ephemeral existing somewhere in between these two points - and as technologies "upgrade" and error-checking protocols horde in, this particular glitch may cease to exist. This ephemeral nature helps us become aware of the complex and fragile digital dynamism at play and assumptions we might often make to the contrary. This hack in particular always exists in the present and never in the past because every use of the "glitchcodec" necessitates a new destruction of a codec[file] at the code level - creation by destruction. The Glitch Codec Tutorial itself can be discussed as glitch art but it's important to note that the Glitch Codec Tutorial itself is a work that is always experienced and never observed - it's a process not a product.

// THANKS

Ben Chang
Nic Collins
& jonCates

++++++++++++++++++++++

Beth Capper
Faith Wilding
Jon Satrom
Rosa Menkman
Evan Meaney
Robb Drinkwater
Jake Elliott
Frédéric Moffet
Chris Sullivan

and everyone who participated in the workshops

++++++++++++++++++++++

//Big Ups

Thomas Jefferson
Yochai Benkler
Phil Morton
Dan Sandin
Richard Stallman
Shirley Clarke
Lawrence Lessig
& Adrian Mackenzie