发布于[2015/12/22](http://www.shenqii.com/?p=256) 由[申 祺](http://www.shenqii.com/?author=3)

**配置 Grbl**

Grbl has a system ‘$’-command to tweak the settings and trigger runtime commands. Connect to Grbl using the serial terminal of your choice (baud rate 9600 unless you changed that in config.h) as 8-N-1 (8-bits, no parity, and 1-stop bit.) Once connected you should get the Grbl-prompt, which looks like this:

Grbl 有一个系统“$” – 命令来调整设置和触发运行命令。使用您所选择的串行终端连接 Grbl（默认 波特率 9600 除非你在 config.h 更改过）8-N-1（8 位，无奇偶校验，1 停止位）一旦连接你会收到 Grbl 提示，就像这样：

Grbl 0.8c ['$' for help]

Type $ and press enter. You should not see any local echo of the $ and enter, but Grbl should respond with a help message:

输入 $ 并按回车键。Grbl 会返回一些帮助信息：

$$ (view Grbl settings)

$# (view # parameters)

$G (view parser state)

$N (view startup blocks)

$x=value (save Grbl setting)

$Nx=line (save startup block)

$C (check gcode mode)

$X (kill alarm lock)

$H (run homing cycle)

~ (cycle start)

! (feed hold)

? (current status)

ctrl-x (reset Grbl)

Of these commands, only four of them are used to configure Grbl, while the rest are runtime commands that either report the current state in Grbl or tweak the behavior.

这些命令中，只有其中的四个用于配置 Grbl，而其余的是运行命令，旨在报告 Grbl 当前状态或调整行为。**这四个配置命令是：**

**[$$, $x=value, $N, $Nx=line]**

**$$ – 查看 Grbl 设置，$X = VAL – 保存 Grbl 设置**

When connected to Grbl, type $$ and press enter. Grbl should respond with a list of the current system settings, similar to as shown in the following. All of these settings are persistent and kept in EEPROM, so if you power down, these will be loaded back up the next time your power up your Arduino.

当连接到 Grbl，输入 $$ 并按回车键。Grbl 便会列出当前系统设置，类似于如下所示。所有这些设置都是永久性的保存在 EEPROM 中，因此，即使你关闭电源，在你下次启动 Arduino 时这些备份同样会被加载。

$0=755.906 (x, step/mm)

$1=755.906 (y, step/mm)

$2=755.906 (z, step/mm)

$3=30 (step pulse, usec)

$4=500.000 (default feed, mm/min)

$5=500.000 (default seek, mm/min)

$6=28 (step port invert mask, int:00011100)

$7=25 (step idle delay, msec)

$8=50.000 (acceleration, mm/sec^2)

$9=0.050 (junction deviation, mm)

$10=0.100 (arc, mm/segment)

$11=25 (n-arc correction, int)

$12=3 (n-decimals, int)

$13=0 (report inches, bool)

$14=1 (auto start, bool)

$15=0 (invert step enable, bool)

$16=0 (hard limits, bool)

$17=0 (homing cycle, bool)

$18=0 (homing dir invert mask, int:00000000)

$19=25.000 (homing feed, mm/min)

$20=250.000 (homing seek, mm/min)

$21=100 (homing debounce, msec)

$22=1.000 (homing pull-off, mm)

To change e.g. the microseconds step pulse option to 10us you would type this, followed by an enter:

比方说要更改 微秒步进脉冲选项为 10μs 您可以参照下面方式输入，然后按回车：

$3=10

If everything went well, Grbl will respond with an ‘ok’ and this setting is stored in EEPROM and will be retained forever or until you change them. You can check if Grbl has received and stored your setting correctly by typing $$ to view the system settings again.

如果一切顺利，Grbl 会提示“OK”，并将此设置永久性保存在 EEPROM 中，直到你再次修改它们。你可以输入 $$ 再次查看系统设置以验证 Grbl 是否已收到并正确保存了你的设置。

**$N – 查看启动区块，$Nx=line – 保存启动区块**

$Nx are the startup blocks that Grbl runs everytime you power on Grbl or reset Grbl. In other words, a startup block is a line of g-code that you can have Grbl auto-magically run to set your g-code defaults, or anything else you need Grbl to do everytime you start up your machine. At the moment, Grbl will store two blocks of g-code as a system default, but it can store anywhere from 1 to 5, as a compile-time option (changing config.h and re-compiling).

So, when connected to Grbl, type $N and then enter. Grbl should respond with something short like:

$Nx 是你每次开机运行 Grbl 或重置 Grbl 的启动块。此时，Grbl 将存储两个 G 代码块作为系统默认值，但它可以存储在从 1 到 5 任何地方，作为一个编译时选项（更改 config.h 并重新编译）。

所以，当连接到 Grbl，键入$N，然后回车。Grbl 会反馈一些短语，如：

$N0=

$N1=

ok

Not much to go on, but this just means that there is no g-code block stored in line $N0 for Grbl to run upon startup. $N1 is the next line to be run. (If you re-compile to have more, it will execute in order on to $N4.)

To set a startup block, type $N0= followed by a valid g-code block and an enter. Grbl will run the block to check if it’s valid and then reply with an ok or an error: to tell you if it’s successful or something went wrong.

For example, say that you want to use your first startup block $N0 to set your g-code parser modes like G54 work coordinate, G20 inches mode, G17 XY-plane. You would type $N0=G20 G54 G17 with an enter and you should see an ‘ok’ response. You can then check if it got stored by typing $N and you should now see a response like $N0=G20G54G17.

Once you have a startup block stored in Grbl’s EEPROM, everytime you startup or reset you will see your startup block printed back to you and a response from Grbl to indicate if it ran ok. So for the previous example, you’ll see:

Grbl 0.8c ['$' for help]

G20G54G17ok

If you have multiple g-code startup blocks, they will print back to you in order upon every startup. And if you’d like to clear one of the startup blocks, type $N0= without anything following the equal sign.

Also, if you have homing enabled, the startup blocks will execute immediately after the homing cycle, not at startup.

IMPORTANT: Be very careful when storing any motion (G0/1,G2/3,G28/30) commands in the startup blocks. These motion commands will run everytime you reset or power up Grbl, so if you have an emergency situation and have to e-stop and reset, a startup block move can and will likely make things worse quickly.

**Grbl’s $x=val 设置及其含义**

**$0, $1 and $2 – XYZ, steps/mm**

Grbl needs to know how far each step will take the tool in reality. To calculate steps/mm for an axis of your machine you need to know:

在实际应用中 Grbl 需要知道每一步将会走多远，为了计算 步数/毫米 你需要知道您机器的轴的螺距：

* The mm per revolution of the lead screw
* The full steps per revolution of your steppers (typically 200)
* The microsteps per step of your controller (typically 1, 2, 4, 8, or 16). *Tip: Using high microstep values (e.g 16) can reduce your stepper motor torque, so use the lowest that gives you the desired axes resolution and comfortable running properties.*

The steps/mm can then be calculated like this:

steps\_per\_mm = (steps\_per\_revolution\*microsteps)/mm\_per\_rev

Compute this value for every axis and write these settings to Grbl.

然后 步数/毫米 可以这样计算：步数 = （360°/步距角） x 细分数 / 丝杠螺距

计算每个轴的数值并将这些设置写入 Grbl。

**$3 – Step pulse, microseconds**

Stepper drivers are rated for a certain minimum step pulse length. Check the data sheet or just try some numbers. You want as short pulses as the stepper drivers can reliably recognize. If the pulses are too long you might run into trouble running the system at high feed rates. Generally something between 5 and 50 microseconds works fine.

步进驱动器额定范围的最小步进脉冲长度。你想作为短脉冲同样能够被步进驱动器可靠地识别。请检查数据表或尝试一些数字。如果脉冲太长，在高进给率运行时您可能会遇到系统故障。一般介于 5 至 50 微秒工作正常。

**$4 and $5 – 默认的进给和寻址率 ， 毫米/分钟**

This setting sets the default seek(G0) and feed rates(G1,G2,G3) after Grbl powers on and initializes. The seek rate (aka rapids) is used for moving from point A to point B as quickly as possible, usually for traversing into position. The seek rate should be set at the maximum speed your machine can go in any axes movement. The default feed rate usually does not enter into the picture as feed rates will generally be specified in the g-code program, but if not, this default feed rate will be used.

这个设置是在 Grbl 加电并初始化后设置默认寻址（G0）和进给率（G1，G2。G3）。此寻址率（即急速）用于从 A 点尽可能快地移动到 B 点，通常为寻迹到的位置。在任何轴运动中，都应该以最大速度设置寻址率。默认的进给速率一般会在特定的 G 代码程序中通常不随着进给率进入图形。但如果没有，便将使用默认的进给速率。

NOTE: In future releases, the default feed rate will be removed. G-code standards do not have a default feed rate (G1,G2,G3), but error out if there has been no ‘F’ feed rate command specified. The seek rate (aka rapids) will be updated so users can specify the maximum speed each axes can move, rather than have just one speed that limits all of them.

注意：在未来的版本中，默认的进给速度将被移除。G 代码标准中没有默认进给速度（G1，G2，G3），如果没有“F”进给速度命令指定，则报错。寻址速度（即急速）将被更新，这样用户可以指定每个轴最大可移动速度。

**$6 – Step port invert mask, int:binary**

Some cnc-stepper controllers needs its high-low inputs inverted for both direction and steps. Signal lines are normally held high or low to signal direction or held high and goes low for a couple of microseconds to signal a step event. To achieve this, Grbl can invert the output bits to accomodate particular needs. The invert mask value is a byte that is xored with the step and direction data before it is sent down the stepping port. That way you can use this both to invert step pulses or to invert one or more of the directions of the axes. The bits in this byte corresponds to the pins assigned to stepping in config.h. Note that bits 0 and 1 are not used for inversion. Per default bits are assigned like this:

#define X\_STEP\_BIT 2

#define Y\_STEP\_BIT 3

#define Z\_STEP\_BIT 4

#define X\_DIRECTION\_BIT 5

#define Y\_DIRECTION\_BIT 6

#define Z\_DIRECTION\_BIT 7

If you wanted to invert the X and Y direction in this setup you would calculate a value by bitshifting like this (in your favorite calculating environment):

> (1<<X\_DIRECTION\_BIT)|(1<<Y\_DIRECTION\_BIT)

Which is equal to 96, so issuing this command would invert them:

$6=96

Now when you view the current settings, you should now see this in your invert mask line with the binary representation of the number (bits 5 and 6 should now show a 1 to indicate inversion.)

$6=96 (step port invert mask. int:1100000)

**$7 – 步空闲延迟，毫秒**

Every time your steppers complete a motion and come to a stop, Grbl will disable the steppers by default. The stepper idle lock time is the time length Grbl will keep the steppers locked before disabling. Depending on the system, you can set this to zero and disable it. On others, you may need 25-50 milliseconds to make sure your axes come to complete stop before disabling. (My machine tends to drift just slightly if I don’t have this enabled.) OR, you can always keep your axes enabled at all times by setting this value to the maximum 255 milliseconds. Again, just to repeat, you can keep all axes always enabled by setting $7=255.

每次你的步进电机完成一个动作，并停下来，Grbl 默认会被禁用步进电机。步进电机空闲锁定时间的时间长短 Grbl 会使步进电机保持锁定之前禁用（这段话逻辑有些乱，还没弄明白什么意思）。 根据系统的不同，您可以将此项设置为零，并禁用它。在其他系统上，你可能需要 25-50 毫秒，以确保您的轴在完全停止之前禁用。 （如果我没有启用，我的机器就有可能稍微漂移。）或者，你可以通过将此值设置为最大 255 毫秒，来始终保持你的轴在任何时候为启用状态。再次，只要重复，通过设置 $7=255 你可以保持所有轴始终启用。

**$8 – 加速度，mm/sec^2**

This is the acceleration in mm/second/second. You don’t have to understand what that means, suffice it to say that a lower value gives smooooother acceleration while a higher value yields tighter moves and reach the desired feedrates much quicker. In technical terms, this is the point to point acceleration of your machine, independent of axes. Set this acceleration value as high as your most limiting axes can let you without losing *ANY* steps. Usually you’d like to give yourself a little buffer, because if you lose steps, Grbl has no idea this has happened (steppers are open-loop control) and will keep going.

**$9 – Junction deviation, mm**

Cornering junction deviation is used by the acceleration manager to determine how fast it can move through a path. The math is a bit complicated but in general, higher values gives generally faster, possibly jerkier motion. Lower values makes the acceleration manager more careful and will lead to careful and slower cornering. So if you run into problems where your machine tries to take a corner too fast, *decrease* this value to make it slow down. If you want your machine to move faster through junctions, *increase* this value to speed it up. For technical people, hit this [link](http://t.co/KQ5BvueY) to read about Grbl’s cornering algorithm, which accounts for both velocity and junction angle with a very simple, efficient, and robust method.

**$10 – 弧， mm/segment**

Grbl renders circles and arcs by subdividing them into teeny tiny lines. You will probably never need to adjust this value – but if you find that your circles are too crude (really? one tenth of a millimeter is not precise enough for you? Are you in nanotech?) you may adjust this. Lower values gives higher precision but may lead to performance issues.

**$11 – N-arc correction, int**

This is an advanced setting that shouldn’t be changed unless there are circumstances that you need to. To make G02/03 arcs possible in Grbl, Grbl approximates the location of the next arc segment by a small angle approximation. N-arc correction is the number of approximate arc segments performed before Grbl computes an exact arc segment to correct for the approximation error drift. Computing these exact locations are computationally expensive, but there are some extreme cases where the small angle approximations can introduce enough error to be noticeable, like very very small arcs with a large arc segment length. Change this setting only if find strange problems with arcs, but it’s not recommended to go below 3 because this may lead to buffer starvation, where the axes slow down and hiccup. But, results can vary.

**$12 – N-decimal, int**

Set how many decimal places all of the floating point values Grbl reports. Not much more complicated than that.

设置 Grbl 报告的所有浮点值小数点后多少位。没什么比这更复杂。

**$13 – Report inches, bool**

Grbl v0.8 has a real-time positioning reporting feature to provide a user feedback on where the machine is exactly at that time. By default it is set to report in mm, but by sending a $13=1 command, you send this boolean flag to true and the status reporting feature will now report in inches. $13=0 to set back to mm.

**$14 – 自动启动，布尔**

In a more professional CNC environment, pros start a job by loading up their program and then pressing the ‘cycle start’ button on their machine. It begins the job. Grbl does the same thing, but not by default. As a learning tool, we ‘auto-cycle start’ any g-code command that user sends to Grbl, for just jogging the machine to see if move in the direction they think it should go, or just seeing with what their machine can do. This makes it easier to load up your machine with Grbl and pick up on how it works, rather than having to diligently hit ‘cycle start’ anytime you want to move any your axes. Once you learn your machine and get a good handle on g-code, you can disable the ‘auto cycle-start’ feature by sending Grbl $14=0 command. (May need a soft-reset or power cycle to load the change.)

Another way of saying that might be:

If $14=0 then some gcode commands like Xn, Yn etc. won’t happen when you enter them at a serial terminal. If $14=1 then the gcode motion commands will happen.

Apparently big cnc’s won’t execute gcode until the operator presses a ‘cycle start’ button. The gcode commands were received but queued and awaiting the ‘cycle start’ button. Makes sense yes? When $14=0 then that’s how grbl acts too. You need the button! (in which case attach the feed-hold button too!).

When $14=1 then the grbl software automatically presses a software version of the ‘cycle start’ for you each time you ‘enter’ a gcode command line via a serial terminal. This is done to make it more convenient for you to enter commands and see something happen without having to push the button.

**$15 – 反转 步 启用，布尔**

By default, the stepper enable pin is high to disable and low to enable. If your setup needs the opposite, just invert the stepper enable pin by typing $15=1. Disable with $15=0. (May need a power cycle to load the change.)

默认情况下，步进使能引脚是高电平为禁用和低电平为启用。如果您需要相反的设置，反转步进使能引脚输入 $15=1。禁用为 $15=0。（可能需要一个动力循环加载的变化。）

**$16 – 硬 限位，布尔**

Hard limit switches are a safety feature to help prevent your machine from traveling way too far off the ends of travel and crashing or breaking something expensive. Basically you wire up some switches (mechanical or optical) near the end of travel of each axes, or where ever you feel like there might be trouble if your program moves too far to where it shouldn’t. When the switch triggers, it will immediately stop all motion, shutdown the coolant and spindle (if connected), and go into alarm mode, which forces you to check your machine and reset everything.

硬限位开关是一项安全特性，用来防止您的机器超出行程发生碰撞造成严重损失。总的说来，你应该在靠近每个轴的行程末端连接一些开关（机械或光学 的），或许你会觉得太麻烦，因为你觉得你的程序应该移动不到那么远的地方。当开关触发，它将立即停止所有动作，关闭冷却液和主轴（假如主轴已连接），并进 入报警模式，这迫使你不得不检查你的机器还得重置一切。

To use hard limits with Grbl, the limit pins are held high with an internal pull-up resistor, so all you have to do is wire in a normally-open switch with the pin and ground and enable hard limits with $16=1. That’s it. (Disable with $16=0) If you want a limit for both ends of travel of one axes, just wire in two switches in parallel with the pin and ground, so if either one of them trips, it triggers the hard limit.

若要使用 Grbl 的硬限制，限位引脚内部已有保持高位的上拉电阻，所以，你要做的就是连接一个常开开关到引脚并接地，然后，启用硬限制 $16=1。（禁用是 $16 = 0），就是这样。如果你想使用一个开关限制轴的两端行程，只需并联连接两个开关到引脚并接地，这样如果其中任何一个跳闸，都会触发硬限制。

Just know, that a hard limit event is considered to be critical event, where steppers immediately stop and will have likely lost steps. Grbl doesn’t have any feedback on position, so it can’t guarantee it has any idea where it is. So, if a hard limit is triggered, Grbl will go into an infinite loop ALARM mode, giving you a chance to check your machine and forcing you to reset Grbl. Remember it’s a purely a safety feature.

要知道，一个硬限制事件被认为是重要事件，其中步进电机立即停止，还将有可能丢失步数。Grbl 不会对位置作任何反馈，所以它也无能为力。因此，如果一个硬性限制被触发，Grbl 将进入无限循环报警模式，让您有机会来检查你的机器，并迫使你重置 Grbl。请记住这是一项纯粹的安全功能。

If you have issues with the hard limit switch constantly triggering after you reset, a soft-reset will reset Grbl into an alarm state, where you can access the settings and Grbl commands, but all g-codes and startup blocks will be locked out. So you can disable the hard limits setting and then $X unlock the alarm. Or, you can wire in a normally-closed switch in series with ground to all the limit switches to disconnect the switches temporarily so you can have Grbl move your axes off the switches.

如果你在重置后有硬限位开关不断触发的问题，软复位后，Grbl 进入报警状态，在这里您可以访问设置和 Grbl 命令，但所有的 G 代码和启动区块将被锁定。这时你可以禁用硬限制设置，然后输入 $X 解除警报。或者，您可以给所有的限位开关串联一个常闭开关并接地，暂时断开开关，以便你有办法关掉传感器并移动你的轴。

**$17 – Homing cycle, bool**

Ahh, homing. Something that has been sorely needed in Grbl for a long time. It’s now fully supported in v0.8. For those just initiated into CNC, the homing cycle is used to accurately and precisely locate position zero on a machine (aka machine zero) everytime you startup your Grbl between sessions. In other words, you know exactly where you are at any given time, every time. Say you start machining something or are about to start the next step in a job and the power goes out, you re-start Grbl and Grbl has no idea where it is. You’re left with the task of figuring out where you are. If you have homing, you always have the machine zero reference point to locate from, so all you have to do is run the homing cycle and resume where you left off.

To set up the homing cycle for Grbl, you need to have limit switches in a fixed position that won’t get bumped or moved, or else your reference point gets messed up. Usually they are setup in the farthest point in +x, +y, +z of each axes. Wire your limit switches in with the limit pins and ground, just like with the hard limits, and enable homing. If you’re curious, you can use your limit switches for both hard limits AND homing. They play nice with each other.

By default, Grbl’s homing cycle moves the Z-axis positive first to clear the workspace and then moves both the X and Y-axes at the same time in the positive direction. To set up how your homing cycle behaves, there are more Grbl settings down the page describing what they do (and compile-time options as well.)

Also, one more thing to note, when homing is enabled. Grbl will lock out all g-code commands until you perform a homing cycle. Meaning no axes motions, unless the lock is disabled ($X) but more on that later. Most, if not all CNC controllers, do something similar, as it mostly a safety feature to help users from making positioning mistake, which is very easy to do and be saddening when a mistake ruins a part. If you find this annoying or find any weird bugs, please let us know and we’ll try to work on it so everyone is happy.

NOTE: Check out config.h for more homing options for advanced users. You can disable the homing lockout at startup, configure which axes move first during a homing cycle and in what order, and more.

**$18 – Homing dir invert mask, int:binary**

By default, Grbl assumes your homing limit switches are in the positive direction, first moving the z-axis positive, then the x-y axes positive before trying to precisely locate machine zero by going back and forth slowly around the switch. If your machine has a limit switch in the negative direction, the homing direction mask can invert the axes direction. It works just like the invert stepper mask, where all you have to do set the axis direction pins to 1 that you want to invert and that axes will now search for the limit pin in the negative direction.

**$19 – Homing feed, mm/min**

The homing cycle first searches for the limit switches at a higher seek rate, and after it finds them, it moves at a slower feed rate to hone into the precise location of machine zero. Homing feed rate is that slower feed rate. Set this to whatever rate value that provides repeatable and precise machine zero locating.

**$20 – Homing seek, mm/min**

Homing seek rate is the homing cycle search rate, or the rate at which it first tries to find the limit switches. Adjust to whatever rate gets to the limit switches in a short enough time without crashing into your limit switches if they come in too fast. This seek rate behaves a little differently than the main stepper driver. Instead of the rate from point to point, it just moves all of the axes at the same individual rate, regardless of how many axes are moving at the same time. So, the XY seek move will seem to move about 41% faster than if you would move it with a G1 command. (You can disable this in config.h if it bothers you. It’s there to speed up the homing cycle.)

**$21 – Homing debounce, ms**

Whenever a switch triggers, some of them can have electrical/mechanical noise that actually ‘bounce’ the signal high and low for a few milliseconds before settling in. To solve this, you need to debounce the signal, either by hardware with some kind of signal conditioner or by software with a short delay to let the signal finish bouncing. Grbl performs a short delay only homing when locating machine zero. Set this delay value to whatever your switch needs to get repeatable homing. In most cases, 5-25 milliseconds is fine.

**$22 – Homing pull-off, mm**

To play nice with the hard limits feature, where homing can share the same limit switches, the homing cycle will move off the all of the limit switches by this pull-off travel after it completes. In other words, it helps to prevent accidental triggering of the hard limit after a homing cycle.

The homing seek rate setting controls how quickly the pull-off maneuver moves, like a G1 command.