Grbl v1.1 Commands

Based on: https://github.com/gnea/grbl/wiki/Grbl-v1.1-Commands by Thomas Moyer

In general, Grbl assumes all characters and streaming data sent to it is g-code and will parse and try to execute it as soon as it can. However, Grbl also has two separate system command types that are outside of the normal g-code streaming. One system command type is streamed to Grbl like g-code, but starts with a $ character to tell Grbl it's not g-code. The other is composed of a special set of characters that will immediately command Grbl to do a task in real-time. It's not part of the g-code stream. Grbl's system commands do things like control machine state, report saved parameters or what Grbl is doing, save or print machine settings, run a homing cycle, or make the machine move faster or slower than programmed. This document describes these "internal" system Grbl commands, what they do, how they work, and how to use them.

Getting Started

First, connect to Grbl using the serial terminal of your choice.

Set the baud rate to **115200** 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 1.1e ['$' for help]
```

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

```
[HLP:$$ $# $G $I $N $x=val $Nx=line $J=line $SLP $C $X $H ~ ! ? ctrl-x]
```

The '$'-commands are Grbl system commands used to tweak the settings, view or change Grbl's states and running modes, and start a homing cycle. The last four **non**-'$' commands are realtime control commands that can be sent at anytime, no matter what Grbl is doing. These either immediately change Grbl's running behavior or immediately print a report of the important realtime data like current position (aka DRO). There are over a dozen more realtime control commands, but they are not user type-able. See realtime command section for details.

Grbl '$' Commands

The $ system commands provide additional controls for the user, such as printing feedback on the current G-code parser modal state or running the homing cycle. This section explains what these commands are and how to use them.

$$and $x=val - View and write Grbl settings

See Grbl v1.1 Configuration for more details on how to view and write setting and learn what they are.

$# - View gcode parameters

G-code parameters store the coordinate offset values for G54-G59 work coordinates, G28/G30 pre-defined positions, G92 coordinate offset, tool length offsets, and probing (not officially, but we added here anyway). Most of these parameters are directly written to EEPROM anytime they are changed and are persistent. Meaning that they will remain the same, regardless of power-down, until they are explicitly changed. The non-persistent parameters, which will are not retained when reset or power-cycled, are G92, G43.1 tool length offsets, and the G38.2 probing data.

G54-G59 work coordinates can be changed via the G10 L2 Px or G10 L20 Px command defined by the NIST gcode standard and the EMC2 (linuxcnc.org) standard. G28/G30 pre-defined positions can be changed via the G28.1 and the G30.1 commands, respectively.

When $# is called, Grbl will respond with the stored offsets from machine coordinates for each system as follows. TLO denotes tool length offset (for the default z-axis), and PRB denotes the coordinates of the last probing cycle, where the suffix :1 denotes if the last probe was successful and :0 as not successful.

```
[G54:4.000,0.000,0.000]
[G55:4.000,6.000,7.000]
[G56:0.000,0.000,0.000]
[G57:0.000,0.000,0.000]
[G58:0.000,0.000,0.000]
[G59:0.000,0.000,0.000]
[G28:1.000,2.000,0.000]
[G30:4.000,6.000,0.000]
[G92:0.000,0.000,0.000]
[TLO:0.000]
[PRB:0.000,0.000,0.000:0]
```

$G - View gcode parser state

This command prints all of the active gcode modes in Grbl's G-code parser. When sending this command to Grbl, it will reply with a message starting with an [GC: indicator like:

[GC:G0 G54 G17 G21 G90 G94 M0 M5 M9 T0 S0.0 F500.0]

These active modes determine how the next G-code block or command will be interpreted by Grbl's G-code parser. For those new to G-code and CNC machining, modes sets the parser into a particular state so you don't have to constantly tell the parser how to parse it. These modes are organized into sets called "modal groups" that cannot be logically active at the same time. For example, the units modal group sets whether your G-code program is interpreted in inches or in millimeters.

A short list of the modal groups, supported by Grbl, is shown below, but more complete and detailed descriptions can be found at LinuxCNC's website. The G-code commands in **bold** indicate the default modes upon powering-up Grbl or resetting it.

Modal Group	Member Words
Motion Mode	**G0**, G1, G2, G3, G38.2, G38.3, G38.4, G38.5, G80
Coordinate System Select	**G54**, G55, G56, G57, G58, G59
Plane Select	**G17**, G18, G19
Distance Mode	**G90**, G91
Arc IJK Distance Mode	**G91.1**
Feed Rate Mode	G93, **G94**
Units Mode	G20, **G21**
Cutter Radius Compensation	**G40**
Tool Length Offset	G43.1, **G49**
Program Mode	**M0**, M1, M2, M30
Spindle State	M3, M4, **M5**
Coolant State	M7, M8, **M9**

In addition to the G-code parser modes, Grbl will report the active T tool number, S spindle speed, and F feed rate, which all default to 0 upon a reset. For those that are curious, these don't quite fit into nice modal groups, but are just as important for determining the parser state.

Note that this list does not include the **non-modal** g-code commands group and they are not listed in the $G parser report, because they only affect the current line they are commanded in. For completeness, here are the non-modal commands supported by Grbl:

Supported Non-Modal Commands

G4, G10 L2, G10 L20, G28, G30, G28.1, G30.1, G53, G92, G92.1

$I - View build info

This prints feedback to the user the Grbl version and source code build date. Optionally, $I can also store a short string to help identify which CNC machine you are communicating with, if you have more than machine using Grbl. To set this string, send Grbl $I=xxx, where xxx is your customization string that is less than 80 characters. This string will be saved as capitalized, white space removed, and can only contain alpha-numeric characters. The next time you query Grbl with a $I view build info, Grbl will print this string after the version and build date.

NOTE: Some OEMs may block access to over-writing the build info string so they can store product information and codes there.

$N - View startup blocks

$Nx are the startup blocks that Grbl runs every time 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 modal defaults, or anything else you need Grbl to do everytime you start up your machine. Grbl can store two blocks of G-code as a system default.

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

```
$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.

$Nx=line - Save startup block

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. Also, do not place any commands that save data to EEPROM, such as G10/G28.1/G30.1. This will cause Grbl to constantly re-write this data upon every startup and reset, which will eventually wear out your Arduino's EEPROM.

Typical usage for a startup block is simply to set your preferred modal states, such as G20 inches mode, always default to a different work coordinate system, or, to provide a way for a user to run some user-written unique feature that they need for their crazy project.

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. If there is an error, Grbl will not save it.

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, starting with an open-chevron >, and a :ok response from Grbl to indicate if it ran okay. So for the previous example, you'll see:

```
Grbl 1.1d ['$' for help]
>G20G54G17:ok
```

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, (e.g., block 0) type $N0= without anything following the equal sign.

NOTE: There are two variations on when startup blocks will not run. First, it will not run if Grbl initializes up in an ALARM state or exits an ALARM state via an $X unlock for safety reasons. Always address and cancel the ALARM and then finish by a reset, where the startup blocks will run at initialization. Second, if you have homing enabled, the startup blocks will execute immediately after a successful homing cycle, not at startup.

$C - Check gcode mode

This toggles the Grbl's gcode parser to take all incoming blocks and process them completely, as it would in normal operation, but it does not move any of the axes, ignores dwells, and powers off the spindle and coolant. This is intended as a way to provide the user a way to check how their new G-code program fares with Grbl's parser and monitor for any errors (and checks for soft limit violations, if enabled).

When toggled off, Grbl will perform an automatic soft-reset (^X). This is for two purposes. It simplifies the code management a bit. But, it also prevents users from starting a job when their G-code modes are not what they think they are. A system reset always gives the user a fresh, consistent start.

$X - Kill alarm lock

Grbl's alarm mode is a state when something has gone critically wrong, such as a hard limit or an abort during a cycle, or if Grbl doesn't know its position. By default, if you have homing enabled and power-up the Arduino, Grbl enters the alarm state, because it does not know its position. The alarm mode will lock all G-code commands until the '$H' homing cycle has been performed. Or if a user needs to override the alarm lock to move their axes off their limit switches, for example, '$X' kill alarm lock will override the locks and allow G-code functions to work again.

But, tread carefully!! This should only be used in emergency situations. The position has likely been lost, and Grbl may not be where you think it is. So, it's advised to use G91 incremental mode to make short moves. Then, perform a homing cycle or reset immediately afterwards.

As noted earlier, startup lines do not execute after a $X command. Always reset when you have cleared the alarm and fixed the scenario that caused it. When Grbl resets to idle, the startup lines will then run as normal.

$H - Run homing cycle

This command is the only way to perform the homing cycle in Grbl. Some other motion controllers designate a special G-code command to run a homing cycle, but this is incorrect according to the G-code standards. Homing is a completely separate command handled by the controller.

TIP: After running a homing cycle, rather jogging manually all the time to a position in the middle of your workspace volume. You can set a G28 or G30 pre-defined position to be your post-homing position, closer to where you'll be machining. To set these, you'll first need to jog your machine to where you would want it to move to after homing. Type G28.1 (or G30.1) to have Grbl store that position. So then after '$H' homing, you could just enter 'G28' (or 'G30') and it'll move there auto-magically. In general, I would just move the XY axis to the center and leave the Z-axis up. This ensures that there isn't a chance the tool in the spindle will interfere and that it doesn't catch on anything.

$J=line - Run jogging motion

New to Grbl v1.1, this command will execute a special jogging motion. There are three main differences between a jogging motion and a motion commanded by a g-code line.

- Like normal g-code commands, several jog motions may be queued into the planner buffer, but the jogging can be easily canceled by a jog-cancel or feed-hold real-time command. Grbl will immediately hold the current jog and then automatically purge the buffers of any remaining commands.
- Jog commands are completely independent of the g-code parser state. It will not change any modes like G91 incremental distance mode. So, you no longer have to make sure that you change it back to G90 absolute distance mode afterwards. This helps reduce the chance of starting with the wrong g-code modes enabled.
- If soft-limits are enabled, any jog command that exceeds a soft-limit will simply return an error. It will not throw an alarm as it would with a normal g-code command. This allows for a much more enjoyable and fluid GUI or joystick interaction.

Executing a jog requires a specific command structure, as described below:

- The first three characters must be '$J=' to indicate the jog.
- The jog command follows immediate after the '=' and works like a normal G1 command.
- Feed rate is only interpreted in G94 units per minute. A prior G93 state is ignored during jog.
- Required words:
    - XYZ: One or more axis words with target value.
    - F - Feed rate value. NOTE: Each jog requires this value and is not treated as modal.
- Optional words: Jog executes based on current G20/G21 and G90/G91 g-code parser state. If one of the following optional words is passed, that state is overridden for one command only.
    - G20 or G21 - Inch and millimeter mode
    - G90 or G91 - Absolute and incremental distances
    - G53 - Move in machine coordinates
- All other g-codes, m-codes, and value words are not accepted in the jog command.

- Spaces and comments are allowed in the command. These are removed by the pre-parser.
- Example: G21 and G90 are active modal states prior to jogging. These are sequential commands.
    - $J=X10.0 Y-1.5 F100 will move to X=10.0mm and Y=-1.5mm in work coordinate frame (WPos) at a feed rate of 100.
    - $J=G91 G20 X0.5 F10 will move +0.5 inches (12.7mm) to X=22.7mm (WPos) at a feed rate of 10. Note that G91 and G20 are only applied to this jog command.
    - $J=G53 Y5.0 F10 will move the machine to Y=5.0mm in the machine coordinate frame (MPos) at a feed rate of 10. If the work coordinate offset for the y-axis is 2.0mm, then Y is 3.0mm in (WPos).

Jog commands behave almost identically to normal g-code streaming. Every jog command will return an 'ok' when the jogging motion has been parsed and is setup for execution. If a command is not valid or exceeds a soft-limit, Grbl will return an 'error:'. Multiple jogging commands may be queued in sequence.

NOTE: See jogging documentation for details on using this command to create a low-latency joystick or rotary dial interface.

$RST=$, $RST=#, and $RST=*- Restore Grbl settings and data to defaults

These commands are not listed in the main Grbl $ help message, but are available to allow users to restore parts of or all of Grbl's EEPROM data. Note: Grbl will automatically reset after executing one of these commands to ensure the system is initialized correctly.

- $RST=$: Erases and restores the $$ Grbl settings back to defaults, which is defined by the default settings file used when compiling Grbl. Often OEMs will build their Grbl firmwares with their machine-specific recommended settings. This provides users and OEMs a quick way to get back to square-one, if something went awry or if a user wants to start over.
- $RST=# : Erases and zeros all G54-G59 work coordinate offsets and G28/30 positions stored in EEPROM. These are generally the values seen in the $# parameters printout. This provides an easy way to clear these without having to do it manually for each set with a G20 L2/20 or G28.1/30.1 command.
- $RST=* : This clears and restores all of the EEPROM data used by Grbl. This includes $$ settings, $# parameters, $N startup lines, and $I build info string. Note that this doesn't wipe the entire EEPROM, only the data areas Grbl uses.

To do a complete wipe, please use the Arduino IDE's EEPROM clear example project.

NOTE: Some OEMs may restrict some or all of these commands to prevent certain data they use from being wiped.

$SLP - Enable Sleep Mode

This command will place Grbl into a de-powered sleep state, shutting down the spindle, coolant, and stepper enable pins and block any commands. It may only be exited by a soft-reset or power-cycle. Once re-initialized, Grbl will automatically enter an ALARM state, because it's not sure where it is due to the steppers being disabled.

This feature is useful if you need to automatically de-power everything at the end of a job by adding this command at the end of your g-code program, BUT, it is highly recommended that you add commands to first move your machine to a safe parking location prior to this sleep command. It also should be emphasized that you should have a reliable CNC machine that will disable everything when its supposed to, like your spindle. Grbl is not responsible for any damage it may cause. It's never a good idea to leave your machine unattended. So, use this command with the utmost caution!

Grbl v1.1 Realtime commands

Realtime commands are single control characters that may be sent to Grbl to command and perform an action in real-time. This means that they can be sent at anytime, anywhere, and Grbl will immediately respond, regardless of what it is doing at the time. These commands include a reset, feed hold, resume, status report query, and overrides (in v1.1).

A realtime command:

- Will execute within tens of milliseconds.
- Is a single character that may be sent to Grbl at any time.
- Does not require a line feed or carriage return after them.
- Is not considered a part of the streaming protocol.
- Are intercepted when they are received and never placed in a buffer to be parsed by Grbl.

- Will ignore multiple commands until it has executed the first received command.
- May be tied to an input pin and may be operated with a button or switch.
- Actions depends on state or what Grbl is doing. It may not do anything.
- Descriptions explain how they work and what to expect.

ASCII Realtime Command Descriptions

Four realtime commands are type-able by users on a keyboard and shown in the $ Grbl help message. These realtime command characters control some of Grbl's basic functions.

- 0x18 (ctrl-x) : Soft-Reset
    - Immediately halts and safely resets Grbl without a power-cycle.
    - Accepts and executes this command at any time.
    - If reset while in motion, Grbl will throw an alarm to indicate position may be lost from the motion halt.
    - If reset while not in motion, position is retained and re-homing is not required.
    - An input pin is available to connect a button or switch.
- ? : Status Report Query
    - Immediately generates and sends back runtime data with a status report.
    - Accepts and executes this command at any time, except during a homing cycle and when critical alarm (hard/soft limit error) is thrown.
- ~ : Cycle Start / Resume
    - Resumes a feed hold, a safety door/parking state when the door is closed, and the M0 program pause states.
    - Command is otherwise ignored.
    - If the parking compile-time option is enabled and the safety door state is ready to resume, Grbl will re-enable the spindle and coolant, move back into position, and then resume.
    - An input pin is available to connect a button or switch.
- ! : Feed Hold
    - Places Grbl into a suspend or HOLD state. If in motion, the machine will decelerate to a stop and then be suspended.
    - Command executes when Grbl is in an IDLE, RUN, or JOG state. It is otherwise ignored.

    - If jogging, a feed hold will cancel the jog motion and flush all remaining jog motions in the planner buffer. The state will return from JOG to IDLE or DOOR, if was detected as ajar during the active hold.
    - By machine control definition, a feed hold does not disable the spindle or coolant. Only motion.
    - An input pin is available to connect a button or switch.

Extended-ASCII Realtime Command Descriptions

Grbl v1.1 installed more than a dozen new realtime commands to control feed, rapid, and spindle overrides. To help prevent users from inadvertently altering overrides with a keystroke and allow for more commands later on, all of the new control characters have been moved to the extended ASCII character set. These are not easily type-able on a keyboard, but, depending on the OS, they may be entered using specific keystroke and code. GUI developers will need to be able to send extended ASCII characters, values 128 (0x80) to 255 (0xFF), to Grbl to take advantage of these new features.

- 0x84 : Safety Door
    - Although typically connected to an input pin to detect the opening of a safety door, this command allows a GUI to enact the safety door behavior with this command.
    - Immediately suspends into a DOOR state and disables the spindle and coolant. If in motion, the machine will decelerate to a stop and then be suspended.
    - If executed during homing, Grbl will instead halt motion and throw a homing alarm.
    - If already in a suspend state or HOLD, the DOOR state supersedes it.
    - If the parking compile-time option is enabled, Grbl will park the spindle to a specified location.
    - Command executes when Grbl is in an IDLE, HOLD, RUN, HOMING, or JOG state. It is otherwise ignored.
    - If jogging, a safety door will cancel the jog and all queued motions in the planner buffer. When the safety door is closed and resumed, Grbl will return to an IDLE state.
    - An input pin is available to connect a button or switch, if enabled with a compile-time option.
    - Some builds of Grbl v0.9 used the @ character for this command, but it was undocumented. Moved to extended-ASCII to prevent accidental commanding.

- 0x85 : Jog Cancel
    - Immediately cancels the current jog state by a feed hold and automatically flushing any remaining jog commands in the buffer.
    - Command is ignored, if not in a JOG state or if jog cancel is already invoked and in-process.
    - Grbl will return to the IDLE state or the DOOR state, if the safety door was detected as ajar during the cancel.
- Feed Overrides
    - Immediately alters the feed override value. An active feed motion is altered within tens of milliseconds.
    - Does not alter rapid rates, which include G0, G28, and G30, or jog motions.
    - Feed override value can not be 10% or greater than 200%.
    - If feed override value does not change, the command is ignored.
    - Feed override range and increments may be changed in config.h.
    - The commands are:
        - 0x90 : Set 100% of programmed rate.
        - 0x91 : Increase 10%
        - 0x92 : Decrease 10%
        - 0x93 : Increase 1%
        - 0x94 : Decrease 1%
- Rapid Overrides
    - Immediately alters the rapid override value. An active rapid motion is altered within tens of milliseconds.
    - Only effects rapid motions, which include G0, G28, and G30.
    - If rapid override value does not change, the command is ignored.
    - Rapid override set values may be changed in config.h.
    - The commands are:
        - 0x95 : Set to 100% full rapid rate.
        - 0x96 : Set to 50% of rapid rate.
        - 0x97 : Set to 25% of rapid rate.
- Spindle Speed Overrides
    - Immediately alters the spindle speed override value. An active spindle speed is altered within tens of milliseconds.
    - Override values may be changed at any time, regardless of if the spindle is enabled or disabled.
    - Spindle override value can not be 10% or greater than 200%

        - If spindle override value does not change, the command is ignored.
        - Spindle override range and increments may be altered in config.h.
        - The commands are:
            - 0x99 : Set 100% of programmed spindle speed
            - 0x9A : Increase 10%
            - 0x9B : Decrease 10%
            - 0x9C : Increase 1%
            - 0x9D : Decrease 1%
- 0x9E : Toggle Spindle Stop
    - Toggles spindle enable or disable state immediately, but only while in the HOLD state.
    - The command is otherwise ignored, especially while in motion. This prevents accidental disabling during a job that can either destroy the part/machine or cause personal injury. Industrial machines handle the spindle stop override similarly.
    - When motion restarts via cycle start, the last spindle state will be restored and wait 4.0 seconds (configurable) before resuming the tool path. This ensures the user doesn't forget to turn it back on.
    - While disabled, spindle speed override values may still be altered and will be in effect once the spindle is re-enabled.
    - If a safety door is opened, the DOOR state will supersede the spindle stop override, where it will manage the spindle re-energizing itself upon closing the door and resuming. The prior spindle stop override state is cleared and reset.
- 0xA0 : Toggle Flood Coolant
    - Toggles flood coolant state and output pin until the next toggle or g-code command alters it.
    - May be commanded at any time while in IDLE, RUN, or HOLD states. It is otherwise ignored.
    - This override directly changes the coolant modal state in the g-code parser. Grbl will continue to operate normally like it received and executed an M8 or M9 g-code command.
    - When $G g-code parser state is queried, the toggle override change will be reflected by an M8 enabled or disabled with an M9 or not appearing when M7 is present.
- 0xA1 : Toggle Mist Coolant
    - Enabled by ENABLE_M7 compile-time option. Default is disabled.

- Toggles mist coolant state and output pin until the next toggle or g-code command alters it.
- May be commanded at any time while in IDLE, RUN, or HOLD states. It is otherwise ignored.
- This override directly changes the coolant modal state in the g-code parser. Grbl will continue to operate normally like it received and executed an M7 or M9 g-code command.
- When $G g-code parser state is queried, the toggle override change will be reflected by an M7 enabled or disabled with an M9 or not appearing when M8 is present.