Grbl v1.1 Jogging

This document outlines how to use Grbl v1.1's new jogging commands. These command differ because they can be cancelled and all queued motions are automatically purged with a simple jog-cancel or feed hold real-time command. Jogging command do not alter the g-code parser state in any way, so you no longer have to worry if you remembered to set the distance mode back to `G90` prior to starting a job. Also, jogging works well with an analog joysticks and rotary dials! See the implementation notes below.

How to Use

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

- The first three characters must be '$J=' to indicate the jog.
- The jog command follows immediate after the '=' and works like a normal G1 command.
- Feed rate is only interpreted in G94 units per minute. A prior G93 state is ignored during jog.
- Required words:
  - XYZ: One or more axis words with target value.
  - F - Feed rate value. NOTE: Each jog requires this value and is not treated as modal.
- Optional words: Jog executes based on current G20/G21 and G90/G91 g-code parser state. If one of the following optional words is passed, that state is overridden for one command only.
  - G20 or G21 - Inch and millimeter mode
  - G90 or G91 - Absolute and incremental distances
  - G53 - Move in machine coordinates
  - N line numbers are valid. Will show in reports, if enabled, but is otherwise ignored.
- All other g-codes, m-codes, and value words (including S and T) are not accepted in the jog command.
- Spaces and comments are allowed in the command. These are removed by the pre-parser.
- Example: G21 and G90 are active modal states prior to jogging. These are sequential commands.
  - `$J=X10.0 Y-1.5` will move to X=10.0mm and Y=-1.5mm in work coordinate frame (WPos).
  - `$J=G91 G20 X0.5` will move +0.5 inches (12.7mm) to X=22.7mm (WPos). Note that G91 and G20 are only applied to this jog command.
  - `$J=G53 Y5.0` will move the machine to Y=5.0mm in the machine coordinate frame (MPos). 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, Grbl will return an 'error:'. Multiple jogging commands may be queued in sequence.

The main differences are:

- During a jog, Grbl will report a 'Jog' state while executing the jog.
- A jog command will only be accepted when Grbl is in either the 'Idle' or 'Jog' states.
- Jogging motions may not be mixed with g-code commands while executing, which will return a lockout error, if attempted.
- All jogging motion(s) may be cancelled at anytime with a simple jog cancel realtime command or a feed hold or safety door event. Grbl will automatically flush Grbl's internal buffers of any queued jogging motions and return to the 'Idle' state. No soft-reset required.
- If soft-limits are enabled, jog commands that exceed the machine travel simply does not execute the command and return an error, rather than throwing an alarm in normal operation.
- IMPORTANT: Jogging does not alter the g-code parser state. Hence, no g-code modes need to be explicitly managed, unlike previous ways of implementing jogs with commands like 'G91G1X1F100'. Since G91, G1, and F feed rates are modal and if they are not changed back prior to resuming/starting a job, a job may not run how its was intended and result in a crash.

Joystick Implementation

Jogging in Grbl v1.1 is generally intended to address some prior issues with old bootstrapped jogging methods. Unfortunately, the new Grbl jogging is not a complete solution. Flash and memory restrictions prevent the original envisioned implementation, but most of these can be mimicked by the following suggested methodology.

With a combination of the new jog cancel and moving in `G91` incremental mode, the following implementation can create low latency feel for an analog joystick or similar control device.

- Basic Implementation Overview:
  - Create a loop to read the joystick signal and translate it to a desired jog motion vector.
  - Send Grbl a very short `G91` incremental distance jog command with a feed rate based on the joystick throw.
  - Wait for an 'ok' acknowledgement before restarting the loop.
  - Continually read the joystick input and send Grbl short jog motions to keep Grbl's planner buffer full.
  - If the joystick is returned to its neutral position, stop the jog loop and simply send Grbl a jog cancel real-time command. This will stop motion immediately somewhere along the programmed jog path with virtually zero-latency and automatically flush Grbl's planner queue. It's not advised to use a feed hold to cancel a jog, as it can lead to inadvertently suspending Grbl if its sent after returning to the IDLE state.

The overall idea is to minimize the total distance in the planner queue to provide a low-latency feel to joystick control. The main trick is ensuring there is just enough distance in the planner queue, such that the programmed feed rate is always met. How to compute this will be explain later. In practice, most machines will have a 0.5-1.0 second latency. When combined with the immediate jog cancel command, joystick interaction can be quite enjoyable and satisfying.

However, please note, if a machine has a low acceleration and is being asked to move at a high programmed feed rate, joystick latency can get up to a handful of seconds. It may sound bad, but this is how long it'll take for a low acceleration machine, traveling at a high feed rate, to slow down to a stop. The argument can be made for a low acceleration machine that you really shouldn't be jogging at a high feed rate. It is difficult for a user to gauge where the machine will come to a stop. You risk overshooting your target destination, which can result in an expensive or dangerous crash.

One of the advantages of this approach is that a GUI can deterministically track where Grbl will go by the jog commands it has already sent to Grbl. As long as a jog isn't cancelled, every jog command is guaranteed to execute. In the event a jog cancel is invoked, the GUI would just need to refresh their internal position from a status report after Grbl has cleared planner buffer and returned to the IDLE (or DOOR, if ajar) state from the JOG state. This stopped position will always be somewhere along the programmed jog path. If desired, jogging can then be quickly and easily restarted with a new tracked path.

In combination with `G53` move in machine coordinates, a GUI can restrict jogging from moving into "keep-out" zones inside the machine space. This can be very useful for avoiding crashing into delicate probing hardware, workholding mechanisms, or other fixed features inside machine space that you want to avoid.

How to compute incremental distances

The quickest and easiest way to determine what the length of a jog motion needs to be to minimize latency are defined by the following equations.

`s = v * dt` - Computes distance traveled for next jog command.

where:

- `s` - Incremental distance of jog command.
- `dt` - Estimated execution time of a single jog command in seconds.
- `v` - Current jog feed rate in **mm/sec**, not mm/min. Less than or equal to max jog rate.
- `N` - Number of Grbl planner blocks (`N=15`)
- `T = dt * N` - Computes total estimated latency in seconds.

The time increment `dt` may be defined to whatever value you need. Obviously, you'd like the lowest value, since that translates to lower overall latency `T`. However, it is constrained by two factors.

- `dt > 10ms` - The time it takes Grbl to parse and plan one jog command and receive the next one. Depending on a lot of factors, this can be around 1 to 5 ms. To be conservative, `10ms` is used. Keep in mind that on some systems, this value may still be greater than `10ms` due to round-trip communication latency.
- `dt > v^2 / (2 * a * (N-1))` - The time increment needs to be large enough to ensure the jog feed rate will be acheived. Grbl always plans to a stop over the total distance queued in the planner buffer. This is primarily to ensure the machine will safely stop if a disconnection occurs. This equation simply ensures that `dt` is big enough to satisfy this constraint.
  - For simplicity, use the max jog feed rate for `v` in mm/sec and the smallest acceleration setting between the jog axes being moved in mm/sec^2.
  - For a lower latency, `dt` can be computed for each jog motion, where `v` is the current rate and `a` is the max acceleration along the jog vector. This is very useful if traveling a very slow speeds to locate a part zero. The `v` rate would be much lower in this scenario and the total latency would decrease quadratically.

In practice, most CNC machines will operate with a jogging time increment of `0.025 sec` < `dt` < `0.06 sec`, which translates to about a `0.4` to `0.9` second total latency when traveling at the max jog rate. Good enough for most people.

However, if jogging at a slower speed and a GUI adjusts the `dt` with it, you can get very close to the 0.1 second response time by human-interface guidelines for "feeling instantaneous". Not too shabby!

With some ingenuity, this jogging methodology may be applied to different devices such as a rotary dial or touchscreen. An "inertial-feel", like swipe-scrolling on a smartphone or tablet, can be simulated by managing the jog rate decay and sending Grbl the associated jog commands. While this jogging implementation requires more initial work by a GUI, it is also inherently more flexible because you have complete deterministic control of how jogging behaves.