Search...

# turtle

Turtles are a robotic device, which can break and place blocks, attack mobs, and move about the world. They have an internal inventory of 16 slots, allowing them to store blocks they have broken or would like to place.

## Movement

Turtles are capable of moving through the world. As turtles are blocks themselves, they are confined to Minecraft's grid, moving a single block at a time.

**`turtle.forward`** and **`turtle.back`** move the turtle in the direction it is facing, while **`turtle.up`** and **`turtle.down`** move it up and down (as one might expect!). In order to move left or right, you first need to turn the turtle using **`turtle.turnLeft`**/**`turtle.turnRight`** and then move forward or backwards.

> **ⓘ INFO**
>
> The name "turtle" comes from **Turtle graphics**, which originated from the Logo programming language. Here you'd move a turtle with various commands like "move 10" and "turn left", much like ComputerCraft's turtles!

Moving a turtle (though not turning it) consumes *fuel*. If a turtle does not have any **fuel**, it won't move, and the movement functions will return `false`. If your turtle isn't going anywhere, the first thing to check is if you've fuelled your turtle.

> **HANDLING ERRORS**
>
> Many turtle functions can fail in various ways. For instance, a turtle cannot move forward if there's already a block there. Instead of erroring, functions which can fail either return `true` if they succeed, or `false` and some error message if they fail.
>
> Unexpected failures can often lead to strange behaviour. It's often a good idea to check the return values of these functions, or wrap them in **`assert`** (for instance, use `assert(turtle.forward())` rather than `turtle.forward()`), so the program doesn't misbehave.

## Turtle upgrades

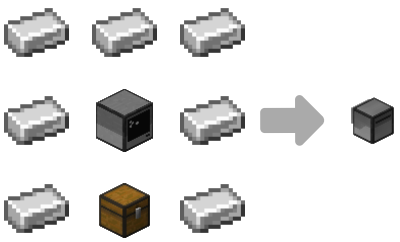
While a normal turtle can move about the world and place blocks, its functionality is limited. Thankfully, turtles can be upgraded with upgrades. Turtles have two upgrade slots, one on the left and right sides. Upgrades can be equipped by crafting a turtle with the upgrade, or calling the `turtle.equipLeft`/`turtle.equipRight` functions.

By default, any diamond tool may be used as an upgrade (though more may be added with **datapacks**). The diamond pickaxe may be used to break blocks (with `turtle.dig`), while the sword can attack entities (`turtle.attack`). Other tools have more niche use-cases, for instance hoes can til dirt.

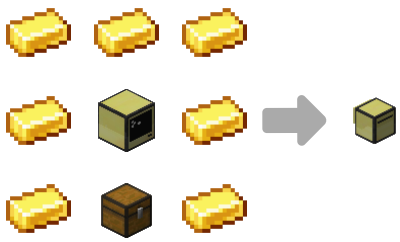
Some peripherals (namely **speakers** and Ender and Wireless **modems**) can also be equipped as upgrades. These are then accessible by accessing the `"left"` or `"right"` peripheral.

## Recipes

**Turtle**



**Advanced Turtle**



### Changes

- **New in version 1.3**

| | |
|---|---|
| `craft([limit=64])` | Craft a recipe based on the turtle's inventory. |
| ~~`native`~~ | The builtin turtle API, without any generated helper functions. |
| `forward()` | Move the turtle forward one block. |
| `back()` | Move the turtle backwards one block. |
| `up()` | Move the turtle up one block. |
| `down()` | Move the turtle down one block. |
| `turnLeft()` | Rotate the turtle 90 degrees to the left. |
| `turnRight()` | Rotate the turtle 90 degrees to the right. |
| `dig([side])` | Attempt to break the block in front of the turtle. |

| | |
|---|---|
| `digUp([side])` | Attempt to break the block above the turtle. |
| `digDown([side])` | Attempt to break the block below the turtle. |
| `place([text])` | Place a block or item into the world in front of the turtle. |
| `placeUp([text])` | Place a block or item into the world above the turtle. |
| `placeDown([text])` | Place a block or item into the world below the turtle. |
| `drop([count])` | Drop the currently selected stack into the inventory in front of the turtle, or as an item into the world if there is no inventory. |
| `dropUp([count])` | Drop the currently selected stack into the inventory above the turtle, or as an item into the world if there is no inventory. |
| `dropDown([count])` | Drop the currently selected stack into the inventory below the turtle, or as an item into the world if there is no inventory. |
| `select(slot)` | Change the currently selected slot. |
| `getItemCount([slot])` | Get the number of items in the given slot. |
| `getItemSpace([slot])` | Get the remaining number of items which may be stored in this stack. |
| `detect()` | Check if there is a solid block in front of the turtle. |
| `detectUp()` | Check if there is a solid block above the turtle. |
| `detectDown()` | Check if there is a solid block below the turtle. |
| `compare()` | Check if the block in front of the turtle is equal to the item in the currently selected slot. |
| `compareUp()` | Check if the block above the turtle is equal to the item in the currently selected slot. |
| `compareDown()` | Check if the block below the turtle is equal to the item in the currently selected slot. |
| `attack([side])` | Attack the entity in front of the turtle. |
| `attackUp([side])` | Attack the entity above the turtle. |

| | |
|---|---|
| `attackDown([side])` | Attack the entity below the turtle. |
| `suck([count])` | Suck an item from the inventory in front of the turtle, or from an item floating in the world. |
| `suckUp([count])` | Suck an item from the inventory above the turtle, or from an item floating in the world. |
| `suckDown([count])` | Suck an item from the inventory below the turtle, or from an item floating in the world. |
| `getFuelLevel()` | Get the maximum amount of fuel this turtle currently holds. |
| `refuel([count])` | Refuel this turtle. |
| `compareTo(slot)` | Compare the item in the currently selected slot to the item in another slot. |
| `transferTo(slot [, count])` | Move an item from the selected slot to another one. |
| `getSelectedSlot()` | Get the currently selected slot. |
| `getFuelLimit()` | Get the maximum amount of fuel this turtle can hold. |
| `equipLeft()` | Equip (or unequip) an item on the left side of this turtle. |
| `equipRight()` | Equip (or unequip) an item on the right side of this turtle. |
| `getEquippedLeft()` | Get the upgrade currently equipped on the left of the turtle. |
| `getEquippedRight()` | Get the upgrade currently equipped on the right of the turtle. |
| `inspect()` | Get information about the block in front of the turtle. |
| `inspectUp()` | Get information about the block above the turtle. |
| `inspectDown()` | Get information about the block below the turtle. |
| `getItemDetail([slot [, detailed]])` | Get detailed information about the items in the given slot. |

## § `craft([limit=64])` [Source]

Craft a recipe based on the turtle's inventory. The turtle's inventory should set up like a crafting grid. For instance, to craft sticks, slots 1 and 5 should contain planks. *All* other slots should be empty, including those outside the crafting "grid".

**Parameters**

1. `limit?` : `number` = 64 The maximum number of crafting steps to run.

**Returns**

1. `true` If crafting succeeds.

Or

1. `false` If crafting fails.
2. **`string`** A string describing why crafting failed.

**Throws**

- When limit is less than 1 or greater than 64.

**Changes**

- **New in version 1.4**

---

## § ~~`native`~~ [Source]

> **ⓘ DEPRECATED**
>
> Historically this table behaved differently to the main turtle API, but this is no longer the case. You should not need to use it.

The builtin turtle API, without any generated helper functions.

---

## § `forward()` [Source]

Move the turtle forward one block.

**Returns**

1. `boolean` Whether the turtle could successfully move.
2. **`string`** | `nil` The reason the turtle could not move.

---

## § `back()` [Source]

Move the turtle backwards one block.

**Returns**

1. `boolean` Whether the turtle could successfully move.
2. **`string`** | `nil` The reason the turtle could not move.

---

## § `up()` [Source]

Move the turtle up one block.

**Returns**

1. `boolean` Whether the turtle could successfully move.
2. **`string`** | `nil` The reason the turtle could not move.

---

## § `down()` [Source]

Move the turtle down one block.

**Returns**

1. `boolean` Whether the turtle could successfully move.
2. **`string`** | `nil` The reason the turtle could not move.

---

## § `turnLeft()` [Source]

Rotate the turtle 90 degrees to the left.

**Returns**

1. `boolean` Whether the turtle could successfully turn.
2. **`string`** | `nil` The reason the turtle could not turn.

---

## § `turnRight()` [Source]

Rotate the turtle 90 degrees to the right.

**Returns**

1. `boolean` Whether the turtle could successfully turn.
2. **`string`** | `nil` The reason the turtle could not turn.

---

## § `dig([side])` [Source]

Attempt to break the block in front of the turtle.

This requires a turtle tool capable of breaking the block. Diamond pickaxes (mining turtles) can break any vanilla block, but other tools (such as axes) are more limited.

**Parameters**

1. `side?` : **`string`** The specific tool to use. Should be "left" or "right".

**Returns**

1. `boolean` Whether a block was broken.
2. **`string`** | `nil` The reason no block was broken.

**Changes**

- **Changed in version 1.6:** Added optional side argument.

---

## § `digUp([side])` [Source]

Attempt to break the block above the turtle. See **`dig`** for full details.

**Parameters**

1. `side?` : **`string`** The specific tool to use.

**Returns**

1. `boolean` Whether a block was broken.
2. **`string`** | `nil` The reason no block was broken.

**Changes**

- **Changed in version 1.6:** Added optional side argument.

---

## § `digDown([side])` [Source]

Attempt to break the block below the turtle. See **`dig`** for full details.

**Parameters**

1. `side?` : **`string`** The specific tool to use.

**Returns**

1. `boolean` Whether a block was broken.
2. **`string`** | `nil` The reason no block was broken.

**Changes**

- **Changed in version 1.6:** Added optional side argument.

## § `place([text])` [Source]

Place a block or item into the world in front of the turtle.

"Placing" an item allows it to interact with blocks and entities in front of the turtle. For instance, buckets can pick up and place down fluids, and wheat can be used to breed cows. However, you cannot use **`place`** to perform arbitrary block interactions, such as clicking buttons or flipping levers.

### Parameters

1. `text? : string` When placing a sign, set its contents to this text.

### Returns

1. `boolean` Whether the block could be placed.
2. **`string`** | `nil` The reason the block was not placed.

### Changes

- **New in version 1.4**

---

## § `placeUp([text])` [Source]

Place a block or item into the world above the turtle.

### Parameters

1. `text? : string` When placing a sign, set its contents to this text.

### Returns

1. `boolean` Whether the block could be placed.
2. **`string`** | `nil` The reason the block was not placed.

### See also

- **`place`** For more information about placing items.

### Changes

- **New in version 1.4**

---

## § `placeDown([text])` [Source]

Place a block or item into the world below the turtle.

### Parameters

1. `text? : string` When placing a sign, set its contents to this text.

**Returns**

1. `boolean` Whether the block could be placed.
2. `string | nil` The reason the block was not placed.

**See also**

- `place` For more information about placing items.

**Changes**

- **New in version 1.4**

---

§ `drop([count])` [Source]

Drop the currently selected stack into the inventory in front of the turtle, or as an item into the world if there is no inventory.

**Parameters**

1. `count? : number` The number of items to drop. If not given, the entire stack will be dropped.

**Returns**

1. `boolean` Whether items were dropped.
2. `string | nil` The reason the no items were dropped.

**Throws**

- If dropping an invalid number of items.

**See also**

- `select`

**Changes**

- **New in version 1.31**

---

§ `dropUp([count])` [Source]

Drop the currently selected stack into the inventory above the turtle, or as an item into the world if there is no inventory.

**Parameters**

1. `count?` : `number` The number of items to drop. If not given, the entire stack will be dropped.

**Returns**

1. `boolean` Whether items were dropped.
2. **`string`** | `nil` The reason the no items were dropped.

**Throws**

- If dropping an invalid number of items.

**See also**

- **`select`**

**Changes**

- **New in version 1.4**

---

## § `dropDown([count])` [Source]

Drop the currently selected stack into the inventory below the turtle, or as an item into the world if there is no inventory.

**Parameters**

1. `count?` : `number` The number of items to drop. If not given, the entire stack will be dropped.

**Returns**

1. `boolean` Whether items were dropped.
2. **`string`** | `nil` The reason the no items were dropped.

**Throws**

- If dropping an invalid number of items.

**See also**

- **`select`**

**Changes**

- **New in version 1.4**

---

## § `select(slot)` [Source]

Change the currently selected slot.

The selected slot is determines what slot actions like **`drop`** or **`getItemCount`** act on.

**Parameters**

1. `slot : number` The slot to select.

**Returns**

1. `true` When the slot has been selected.

**Throws**

- If the slot is out of range.

**See also**

- **`getSelectedSlot`**

---

## § `getItemCount([slot])` [Source]

Get the number of items in the given slot.

**Parameters**

1. `slot? : number` The slot we wish to check. Defaults to the **selected slot**.

**Returns**

1. `number` The number of items in this slot.

**Throws**

- If the slot is out of range.

---

## § `getItemSpace([slot])` [Source]

Get the remaining number of items which may be stored in this stack.

For instance, if a slot contains 13 blocks of dirt, it has room for another 51.

**Parameters**

1. `slot? : number` The slot we wish to check. Defaults to the **selected slot**.

**Returns**

1. `number` The space left in this slot.

**Throws**

- If the slot is out of range.

---

## § `detect()` [Source]

Check if there is a solid block in front of the turtle. In this case, solid refers to any non-air or liquid block.

**Returns**

1. `boolean` If there is a solid block in front.

---

## § `detectUp()` [Source]

Check if there is a solid block above the turtle. In this case, solid refers to any non-air or liquid block.

**Returns**

1. `boolean` If there is a solid block above.

---

## § `detectDown()` [Source]

Check if there is a solid block below the turtle. In this case, solid refers to any non-air or liquid block.

**Returns**

1. `boolean` If there is a solid block below.

---

## § `compare()` [Source]

Check if the block in front of the turtle is equal to the item in the currently selected slot.

**Returns**

1. `boolean` If the block and item are equal.

**Changes**

- **New in version 1.31**

---

## § `compareUp()` [Source]

Check if the block above the turtle is equal to the item in the currently selected slot.

**Returns**

1. `boolean` If the block and item are equal.

**Changes**

- **New in version 1.31**

---

## § `compareDown()` [Source]

Check if the block below the turtle is equal to the item in the currently selected slot.

**Returns**

1. `boolean` If the block and item are equal.

**Changes**

- **New in version 1.31**

---

## § `attack([side])` [Source]

Attack the entity in front of the turtle.

**Parameters**

1. `side?` : **`string`** The specific tool to use.

**Returns**

1. `boolean` Whether an entity was attacked.
2. **`string`** | `nil` The reason nothing was attacked.

**Changes**

- **New in version 1.4**
- **Changed in version 1.6:** Added optional side argument.

---

## § `attackUp([side])` [Source]

Attack the entity above the turtle.

**Parameters**

1. `side?` : **`string`** The specific tool to use.

**Returns**

1. `boolean` Whether an entity was attacked.
2. **`string`** | `nil` The reason nothing was attacked.

**Changes**

- **New in version 1.4**
- **Changed in version 1.6:** Added optional side argument.

---

## § `attackDown([side])` [Source]

Attack the entity below the turtle.

**Parameters**

1. `side?` : **`string`** The specific tool to use.

**Returns**

1. `boolean` Whether an entity was attacked.
2. **`string`** | `nil` The reason nothing was attacked.

**Changes**

- **New in version 1.4**
- **Changed in version 1.6:** Added optional side argument.

---

## § `suck([count])` [Source]

Suck an item from the inventory in front of the turtle, or from an item floating in the world.

This will pull items into the first acceptable slot, starting at the **currently selected** one.

**Parameters**

1. `count?` : `number` The number of items to suck. If not given, up to a stack of items will be picked up.

**Returns**

1. `boolean` Whether items were picked up.
2. **`string`** | `nil` The reason the no items were picked up.

**Throws**

- If given an invalid number of items.

**Changes**

- **New in version 1.4**
- **Changed in version 1.6:** Added an optional limit argument.

## § `suckUp([count])` [Source]

Suck an item from the inventory above the turtle, or from an item floating in the world.

**Parameters**

1. `count?` : `number` The number of items to suck. If not given, up to a stack of items will be picked up.

**Returns**

1. `boolean` Whether items were picked up.
2. **`string`** | `nil` The reason the no items were picked up.

**Throws**

- If given an invalid number of items.

**Changes**

- **New in version 1.4**
- **Changed in version 1.6:** Added an optional limit argument.

---

## § `suckDown([count])` [Source]

Suck an item from the inventory below the turtle, or from an item floating in the world.

**Parameters**

1. `count?` : `number` The number of items to suck. If not given, up to a stack of items will be picked up.

**Returns**

1. `boolean` Whether items were picked up.
2. **`string`** | `nil` The reason the no items were picked up.

**Throws**

- If given an invalid number of items.

**Changes**

- **New in version 1.4**
- **Changed in version 1.6:** Added an optional limit argument.

---

## § `getFuelLevel()` [Source]

Get the maximum amount of fuel this turtle currently holds.

**Returns**

1. `number` The current amount of fuel a turtle this turtle has.

Or

1. `"unlimited"` If turtles do not consume fuel when moving.

**See also**

- `getFuelLimit`
- `refuel`

**Changes**

- **New in version 1.4**

---

## § `refuel([count])` [Source]

Refuel this turtle.

While most actions a turtle can perform (such as digging or placing blocks) are free, moving consumes fuel from the turtle's internal buffer. If a turtle has no fuel, it will not move.

`refuel` refuels the turtle, consuming fuel items (such as coal or lava buckets) from the currently selected slot and converting them into energy. This finishes once the turtle is fully refuelled or all items have been consumed.

**Parameters**

1. `count? : number` The maximum number of items to consume. One can pass `0` to check if an item is combustable or not.

**Returns**

1. `true` If the turtle was refuelled.

Or

1. `false` If the turtle was not refuelled.
2. `string` The reason the turtle was not refuelled.

**Throws**

- If the refuel count is out of range.

**Usage**

- Refuel a turtle from the currently selected slot.

```
local level = turtle.getFuelLevel()
if level == "unlimited" then error("Turtle does not need fuel", 0) end

local ok, err = turtle.refuel()
if ok then
  local new_level = turtle.getFuelLevel()
  print(("Refuelled %d, current level is %d"):format(new_level - level, new_leve
else
  printError(err)
end
```

Run ▷

- Check if the current item is a valid fuel source.

```
local is_fuel, reason = turtle.refuel(0)
if not is_fuel then printError(reason) end
```

Run ▷

**See also**

- `getFuelLevel`
- `getFuelLimit`

**Changes**

- **New in version 1.4**

---

## § `compareTo(slot)` [Source]

Compare the item in the currently selected slot to the item in another slot.

**Parameters**

1. `slot : number` The slot to compare to.

**Returns**

1. `boolean` If the two items are equal.

**Throws**

- If the slot is out of range.

**Changes**

- **New in version 1.4**

---

## § `transferTo(slot [, count])` [Source]

Move an item from the selected slot to another one.

**Parameters**

1. `slot : number` The slot to move this item to.
2. `count? : number` The maximum number of items to move.

**Returns**

1. `boolean` If some items were successfully moved.

**Throws**

- If the slot is out of range.
- If the number of items is out of range.

**Changes**

- **New in version 1.45**

---

## § `getSelectedSlot()` [Source]

Get the currently selected slot.

**Returns**

1. `number` The current slot.

**See also**

- `select`

**Changes**

- **New in version 1.6**

---

## § `getFuelLimit()` [Source]

Get the maximum amount of fuel this turtle can hold.

By default, normal turtles have a limit of 20,000 and advanced turtles of 100,000.

**Returns**

1. `number` The maximum amount of fuel a turtle can hold.

Or

1. `"unlimited"` If turtles do not consume fuel when moving.

**See also**

- `getFuelLevel`
- `refuel`

**Changes**

- **New in version 1.6**

---

## § `equipLeft()` [Source]

Equip (or unequip) an item on the left side of this turtle.

This finds the item in the currently selected slot and attempts to equip it to the left side of the turtle. The previous upgrade is removed and placed into the turtle's inventory. If there is no item in the slot, the previous upgrade is removed, but no new one is equipped.

**Returns**

1. `true` If the item was equipped.

Or

1. `false` If we could not equip the item.
2. `string` The reason equipping this item failed.

**See also**

- `equipRight`
- `getEquippedLeft`

**Changes**

- **New in version 1.6**

---

## § `equipRight()` [Source]

Equip (or unequip) an item on the right side of this turtle.

This finds the item in the currently selected slot and attempts to equip it to the right side of the turtle. The previous upgrade is removed and placed into the turtle's inventory. If there is no item in the slot, the previous upgrade is removed, but no new one is equipped.

**Returns**

1. `true` If the item was equipped.

Or

1. `false` If we could not equip the item.
2. **`string`** The reason equipping this item failed.

**See also**

- **`equipLeft`**
- **`getEquippedRight`**

**Changes**

- **New in version 1.6**

---

## § `getEquippedLeft()` [Source]

Get the upgrade currently equipped on the left of the turtle.

This returns information about the currently equipped item, in the same form as **`getItemDetail`**.

**Returns**

1. **`table`** | `nil` Information about the currently equipped item, or `nil` if no upgrade is equipped.

**See also**

- **`equipLeft`**

**Changes**

- **New in version 1.116.0**

---

## § `getEquippedRight()` [Source]

Get the upgrade currently equipped on the right of the turtle.

This returns information about the currently equipped item, in the same form as **`getItemDetail`**.

**Returns**

1. **`table`** | `nil` Information about the currently equipped item, or `nil` if no upgrade is equipped.

**See also**

- **`equipRight`**

**Changes**

- **New in version 1.116.0**

---

## § inspect() [Source]

Get information about the block in front of the turtle.

### Returns

1. boolean Whether there is a block in front of the turtle.
2. **table** | **string** Information about the block in front, or a message explaining that there is no block.

### Usage

- Run ▷

```lua
local has_block, data = turtle.inspect()
if has_block then
  print(textutils.serialise(data))
  -- {
  --   name = "minecraft:oak_log",
  --   state = { axis = "x" },
  --   tags = { ["minecraft:logs"] = true, ... },
  -- }
else
  print("No block in front of the turtle")
end
```

### Changes

- **New in version 1.64**
- **Changed in version 1.76:** Added block state to return value.

---

## § inspectUp() [Source]

Get information about the block above the turtle.

### Returns

1. boolean Whether there is a block above the turtle.
2. **table** | **string** Information about the above below, or a message explaining that there is no block.

### Changes

- **New in version 1.64**

---

## § inspectDown() [Source]

Get information about the block below the turtle.

**Returns**

1. `boolean` Whether there is a block below the turtle.
2. **`table`** | **`string`** Information about the block below, or a message explaining that there is no block.

**Changes**

- **New in version 1.64**

---

## § `getItemDetail([slot [, detailed]])` [Source]

Get detailed information about the items in the given slot.

**Parameters**

1. `slot?` : `number` The slot to get information about. Defaults to the **selected slot**.
2. `detailed?` : `boolean` Whether to include "detailed" information. When `true` the method will contain much more information about the item at the cost of taking longer to run.

**Returns**

1. `nil` | **`table`** Information about the item in this slot, or `nil` if it is empty.

**Throws**

- If the slot is out of range.

**Usage**

- Print the current slot, assuming it contains 13 dirt.

```
print(textutils.serialise(turtle.getItemDetail()))
-- => {
--  name = "minecraft:dirt",
--  count = 13,
-- }
```

Run ▷

**See also**

- **`inventory.getItemDetail`** Describes the information returned by a detailed query.

**Changes**

- **New in version 1.64**

- **Changed in version 1.90.0:** Added detailed parameter.

Last updated on 2025-07-06