WorldEdit Documentation Documentation

Release 7.4

WorldEdit Team

Jan 18, 2026

CONTENTS

WorldEdit is an easy-to-use in-game Minecraft map editor. Through a combination of commands and "brushes," you can sculpt your world or simply perform numerous terraforming tasks.

- Quickly create, replace or delete thousands of blocks in seconds
- No longer waste time doing mundane activities like fix badly flowing water!
- Quickly create basic shapes like spheres, cylinders, and so on
- Copy areas, paste them, load them, and save them as .schematics
- Do cool things like input mathematical expressions to generate terrain
- Use "brush tools" to carve out mountains, ravines, and so on
- Use your compass to quickly teleport to areas by left clicking or using /jumpto
- Choose an area and have it instantly restored from backups
- Open source, and one of the oldest Minecraft projects (since Minecraft Alpha!)

CHAPTER

ONE

DOCUMENTATION

1.1 Installation

1.1.1 Requirements

WorldEdit runs on the Java edition of Minecraft, either on your single player/local game or a dedicated server.

WorldEdit can't be used on Realms, Windows 10 Edition, Bedrock Edition, or Pocket Edition versions. These versions of Minecraft have limited or no mod support. Note that "Windows 10 Edition" refers to a specific Edition of Minecraft, not the Java Edition running on a Windows 10 computer.

Before you install WorldEdit, you will first have to install a "mod loader" like NeoForge, Fabric, Spigot, Bukkit, or Sponge. We'll advise you to choosing your mod loader below.

Choosing a Mod Loader

If you want to use WorldEdit on your single-player/local game, we recommend one of two choices:

- NeoForge
- Or alternatively, Fabric

On the other hand, if you are running a Minecraft server, you can use

- Paper (recommended over Spigot because it has improvements WorldEdit can use)
- Spigot
- NeoForge (recommended if you are using other NeoForge mods)
- Sponge (also compatible with Forge mods)

Note: Paper and Spigot use its own set of mods (commonly called "plugins") that are mostly incompatible with mods for NeoForge, Fabric, and Sponge. You *may* want to research what other mods/plugins that you may want (generally, Paper and Spigot have much more server administration/"server-ready gameplay mods" - which run completely on the server - and NeoForge, Fabric, and Sponge have more major gameplay mods - which generally require client installation). Regardless, WorldEdit is extremely unique in that it works as a mod for both , so you can just pick the one that seems the easiest and roll with it.

1.1.2 Installation Step-by-step

Bukkit / Spigot / Paper

Once you've set up your Bukkit-based server (instructions can be found on the respective Paper/Spigot sites), download WorldEdit from Modrinth. Make sure you get the right WorldEdit download for your Minecraft version.

1. In your server folder, create a "plugins" folder if one does not yet exist. (It should be created when you first run the server).
2. Move the WorldEdit .jar file into the plugins folder.
3. Start your server.

Check your server log for errors. If you encounter errors, see the *FAQ* page.

NeoForge Single-Player

First, you'll have to install NeoForge. There are many third-party launchers designed to easily install modpacks. If you're using one of those, you can add WorldEdit as a mod through the launcher interface. Otherwise, NeoForge will install a profile available through the official Minecraft Launcher. After installing NeoForge one way or the other, download WorldEdit from Modrinth. Make sure you get the right WorldEdit download for your Minecraft version and platform.

1. If you've installed NeoForge as a profile in the official Minecraft launcher, follow Mojang's instructions for finding where your ".minecraft" folder is. If you're using a third-party launcher, this might be in a different location (consult the launcher's docs).
2. Create a "mods" folder inside the ".minecraft" folder if it doesn't yet exist (it should be created if you've run NeoForge once already).
3. Place the WorldEdit .jar file inside the mods folder. Start NeoForge from your launcher. WorldEdit should show up in the mods list.

If you encounter any errors, see the *FAQ* page.

Fabric Single-Player or Server

First, you'll have to install Fabric. They have instructions on their website depending on how you'd like to install. The MultiMC instructions are recommended for single-player.

Then, download WorldEdit from Modrinth. Make sure you get the right WorldEdit download for your Minecraft version and platform (Forge builds are also there - make sure you grab the right one).

On Minecraft 1.14.x versions, you will also need to install the Fabric API mod. This is not required for 1.15+.

Add the WorldEdit .jar to the "loader mods" section of your MultiMC instance and check to enable it, or add the jar to your mods folder (see NeoForge instructions above) if you're installing in the official Mojang launcher.

NeoForge Server / Sponge

First, you'll have to install your server software of choice. For NeoForge, you can download the installer and run `java -jar neoforge-installer.jar --installServer` from a terminal or command prompt (search online for more comprehensive instructions). Sponge has documentation on setting up a server. After installing your server software, download WorldEdit from Modrinth, if using NeoForge or Ore, if using Sponge.

1. Create a "mods" folder if it doesn't exist (it will be created automatically after running the server once).
2. Add the WorldEdit .jar file to the mods folder.
3. Start the server.

Check your server log for errors. If you encounter errors, see the *FAQ* page.

Want to see selection lines?

To see lines showing your selection, you can either:

1. Access a limited version of the selection outlines server-side via the `//drawsel` command. It works only for cuboid selections that are not larger than 48x48x48 (or 32x32x32 on older versions), and you have to be in creative mode. These limitations are due to how structure blocks have worked in Minecraft for a long time.
2. Use a third party client-side mod, e.g. WorldEdit CUI (Fabric). Note that this mod requires Fabric, so you will have to install it at first.

Note

If you would like to use an older version of Minecraft (1.12 or earlier), in addition to downloading an older WorldEdit (version 6), you may also need the old WorldEditCUI mod by Mumfrey. Note that this mod requires LiteLoader (installation instructions on that page) instead.

1.2 Quick Start

1.2.1 First things first

- If you are using Bukkit, then you need to have permissions to use WorldEdit. You can give yourself op (`/op yourname`) to give yourself permissions. Sponge servers will also need to use permissions, though Sponge does not use op to grant all permissions.
- If you are using NeoForge/Fabric and playing single player, then WorldEdit is only enabled if your world has cheats enabled.
- If you are using NeoForge/Fabric server, then only ops can use WorldEdit.

Tip

For NeoForge/Fabric, you can:

- Turn on "cheat-mode" in WorldEdit's settings (see *Configuration*) file to let you use WorldEdit even in survival (and on a server, everyone is allowed)
- Or instead, turn on "use-in-creative" to let yourself use WorldEdit when you have creative mode (and on a server, when someone has creative)

Want to see selection lines?

To see lines showing your selection, you can either:

1. Access a limited version of the selection outlines server-side via the `//drawsel` command. It works only for cuboid selections that are not larger than 48x48x48 (or 32x32x32 on older versions), and you have to be in creative mode. These limitations are due to how structure blocks have worked in Minecraft for a long time.
2. Use a third party client-side mod, e.g. WorldEdit CUI (Fabric). Note that this mod requires Fabric, so you will have to install it at first.

> **Note**
>
> If you would like to use an older version of Minecraft (1.12 or earlier), in addition to downloading an older WorldEdit (version 6), you may also need the old WorldEditCUI mod by Mumfrey. Note that this mod requires LiteLoader (installation instructions on that page) instead.

The selection lines mod works regardless of how you may have installed WorldEdit (on a Bukkit server, on singleplayer, etc).

1.2.2 Getting Around

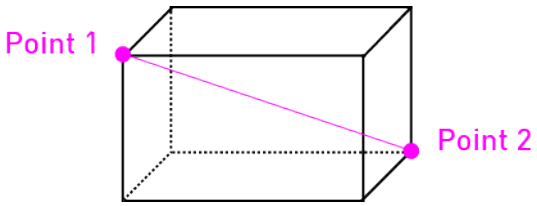
First, let's figure out how you can get around quickly.

1. Look at a block not too far away and type `/jumpto`
2. Stand under a tree and type `/ascend`
3. While on top of the tree, type `/descend`
4. Stand behind a tree trunk, look straight ahead, make sure there's room on the other side, and type `/thru`

Or whip out your compass (or type `//wand -n`), look at a nearby block, and left click. Want to go through walls? Right click on a wall.

1.2.3 Making Selections

A cuboid is like 3D rectangle. In WorldEdit, you select the region that you want by setting two points of a cuboid.



How do you choose the two points? You can either:

- Left and right click blocks while holding a wooden axe (use `//wand` to get a wooden axe)
- Stand somewhere and type `//pos1` and `//pos2`
- Look at a block and type `//hpos1` and `//hpos2`

Tutorial: Make an approximate selection of a 15x15x15 area to test with and go to the next section.

1.2.4 Doing things with the selection

1. Set the entire thing to bedrock: `//set minecraft:bedrock` (`minecraft` is implicit. If you're playing on a platform with mods, you'll need the mod's namespace to identify blocks, eg `//set ic2:stone`.)
2. Set the entire thing to stone: `//set 1` (these are legacy IDs used in Minecraft 1.12 and before. You can use them if you know them already, but it's recommended (and easier) to learn the names of blocks - new blocks in 1.13+ don't have these ids!)
3. Set the selection to 50% sandstone, 50% glass: `//set sandstone,glass`
4. Replace the sandstone with dirt: `//replace sandstone dirt`
5. Clear the area: `//set air`
6. Generate an interesting shape: `//g 35 data=(32+15/2/pi*atan2(x,y))%16; (0.75-sqrt(x^2+y^2))^2+z^2<0.25^2`
7. Look in a cardinal direction (not diagonal) and repeat your selection: `//stack 4`

Let's undo your changes!

- Undo 7 times: `//undo 7`

1.2.5 Adjusting the selection

So you've got a cuboid. Let's change it!

1. Make the cuboid 10 blocks taller, going up: `//expand 10 up`
2. Make the cuboid 5 blocks longer in the direction that you are looking: `//expand 5`
3. Make the cuboid 10 blocks shorter, going down: `//contract 10 down`

1.2.6 Playing with brushes

1. Grab a pickaxe (or any item of choice) and have it as your active slot.
2. Turn on a stone brush of radius 5: `/br sphere stone 5`
3. Aim at ground not near you and right click to place large stone spheres.
4. Make it so the brush only affects grass: `/mask grass`
5. Instead of placing stone, let's place wool: `/material red_wool,green_wool`
6. Right click more areas.
7. Disable the brush: `/brush none`

1.2.7 Continuing on...

Checkout out the rest of the *docs*.

1.3 Configuration

Although WorldEdit is primarily command driven, there are some configuration options which may change or limit the behavior of WorldEdit's commands and tools.

1.3.1 Configuration Files

Once you have run your server with WorldEdit installed, you will find the main configuration file generated in a location which depends on your platform.

Bukkit Server

Configuration options for the Bukkit version of WorldEdit are found in `plugins/WorldEdit/config.yml`, relative to the server root.

Note that the YAML format which Bukkit uses is very sensitive to errors. You must use 4 spaces for indentation (tabs will break the file!), and adhere to YAML's syntax. If you are unfamiliar with editing YAML files, you can run your config through an online validator (like this one) and ensure that it does not return an error.

Forge/Fabric

Configuration options in NeoForge/Fabric can be found in the `config/worldedit/worldedit.properties` file. On a server, this is relative to the server root (where the main server .jar is). On a single-player installation, this is in your ".minecraft" folder.

Sponge Server

Configuration options in Sponge servers (either SpongeForge or SpongeVanilla) can be found in the `config/worldedit/worldedit.conf` file.

1.3.2 Settings

> **Note**
>
> Since the underlying platforms are often in various states of development, not all settings will work on all platforms. Please *notify us* if you find this to be the case.

Setting	Default	Description
defaultLocale	default	The default locale to use for translations, defaults to the system locale.
profile	false	Whether to print out a blocks changed/time info after each operation.
traceUnflushedSessions	false	Display a debug message when an edit isn't completed properly.
disallowedBlocks	<list of blocks>	A list of blocks that cannot be used in patterns (mostly physics blocks that will "pop off" and may severely lag or crash the server if thousands of items are spawned).
defaultChangeLimit	-1	The default amount of blocks that can be set in one operation.
maxChangeLimit	-1	The maximum amount of blocks for the change limit (set with `//limit` in-game)
defaultMaxPolygonal-Points	-1	The default amount of polygonal points that can be used (`//sel poly`), -1 means use maximum
maxPolygonalPoints	20	The maximum amount of polygonal points that can be used (`//sel poly`, used if you have `worldedit.limit.unrestricted`)
defaultMaxPolyhedron-Points	-1	The default amount of polyhedron points that can be used (`//sel convex`), -1 means use maximum
maxPolyhedronPoints	20	The maximum amount of polyhedron points that can be used (`//sel convex`, used if you have `worldedit.limit.unrestricted`)
snapshotRepo		If not left empty, the directory name to look for snapshots
snapshotsExperimental	false	If true, uses the new snapshot code. Try it out and report bugs!

continues on next page

Table 1 – continued from previous page

Setting	Default	Description
maxRadius	-1	Maximum radius of commands that take a radius
maxSuperPickaxeSize	5	Maximum size of super pickaxe tools
maxBrushRadius	6	Maximum size of brushes
logCommands	false	Whether to log more informative information on command usage.
logFile		If logCommands is true, the file to log to.
logFormat	[%1$tY-%1$tm-%1$td %1$tH:%1 %4$s]: %5$s%6$s	The format of command logging
wandItem	minecraft:v	The default item used for selection regions, overriden by existing sessions. Set to -1 for no default.
superPickaxeDrop	true	Whether the single super pickaxe mode will drop items for blocks it breaks
superPickaxeManyDrop	true	Whether multi super pickaxe modes will drop items for blocks they break
useInventory	false	Require players to have items in their inventory to make edits (this option is not well supported and not recommended)
useInventoryOverride	false	Allow a permission node to override the above setting
useInventoryCreativeOverride	false	Allow creative mode to override the above setting
navigationUseGlass	true	Whether the `/up` and `/ceil` commands should place a glass block for the player to stand on if in mid-air
navigationWand	minecraft:c	The default item used for the navigation wand which allows `/jumpto` and `/thru` as left-click/right-click, overriden by existing sessions. Set to -1 for no default.
navigationWandMaxDistance	50	The max distance the navigation wand should trace to find a block to jump to
scriptTimeout	3000	The maximum time a craftscript can run before it is terminated
calculationTimeout	100	The default time an expression can run before it is terminated
maxCalculationTimeout	300	The maximum time an expression can run before termination (changed in game with `//timeout`)
allowedDataCycleBlocks		If not empty, a whitelist of blocks which the data cycler tool can be used on
saveDir	schematics	The directory in which to save schematics (relative to the worldedit folder)
scriptsDir	craftscripts	The directory in which to look for craftscripts
allowSymlinks	false	Whether to allow the above to be symlinked locations (useful for sharing between servers)
butcherDefaultRadius	-1	The default radius of the `/butcher` command (-1 for infinite)
butcherMaxRadius	-1	The maximum radius of the `/butcher` command
serverSideCUI	true	Whether to allow the usage of `//drawsel`
defaultVerticalHeight	256	The height to use for commands that take an optional height.
extendedYLimit	false	If true, use slower but unbounded positions. This should only be needed with a mod that extends the height limit.

1.4 Permissions

By default, no one can use WorldEdit. In order for yourself, moderators, and players to use WorldEdit, you must provide the proper permissions. One way is to provide op to moderators and administrators (unless disabled in the *configuration*), but providing the permission nodes on this page (through a permissions plugin) is the more flexible.

You can give the `worldedit.*` permission to give yourself and other administrators full access to WorldEdit.

1.4.1 Commands

See the *Commands* page for an explanation of some of these commands.

Command	Permission
/worldedit	
/worldedit cui	
/worldedit help	worldedit.help
/worldedit reload	worldedit.reload
/worldedit report	worldedit.report
/worldedit trace	
/worldedit tz	
/worldedit version	
/undo	worldedit.history.undo, worldedit.history.undo.self
/redo	worldedit.history.redo, worldedit.history.redo.self
/clearhistory	worldedit.history.clear
//limit	worldedit.limit
//timeout	worldedit.timeout
//fast	worldedit.fast
//perf	worldedit.perf
//reorder	worldedit.reorder
//drawsel	worldedit.drawsel
//world	worldedit.world
//watchdog	worldedit.watchdog
/gmask	worldedit.global-mask
/toggleplace	
/placement	worldedit.placement
/searchitem	worldedit.searchitem
//registry	worldedit.registry
/unstuck	worldedit.navigation.unstuck
/ascend	worldedit.navigation.ascend
/descend	worldedit.navigation.descend
/ceil	worldedit.navigation.ceiling
/thru	worldedit.navigation.thru.command
/jumpto	worldedit.navigation.jumpto.command
/up	worldedit.navigation.up
//pos	worldedit.selection.pos
//pos1	worldedit.selection.pos
//pos2	worldedit.selection.pos
//hpos1	worldedit.selection.hpos
//hpos2	worldedit.selection.hpos
//chunk	worldedit.selection.chunk
//wand	worldedit.wand
/toggleeditwand	worldedit.wand.toggle
//contract	worldedit.selection.contract

continues on next page

Table 2 – continued from previous page

Command	Permission
//shift	worldedit.selection.shift
//outset	worldedit.selection.outset
//inset	worldedit.selection.inset
//trim	worldedit.selection.trim
//size	worldedit.selection.size
//count	worldedit.analysis.count
//distr	worldedit.analysis.distr
//sel	
//expand	worldedit.selection.expand
//expand vert	
//set	worldedit.region.set
//line	worldedit.region.line
//curve	worldedit.region.curve
//replace	worldedit.region.replace
//overlay	worldedit.region.overlay
//center	worldedit.region.center
//naturalize	worldedit.region.naturalize
//walls	worldedit.region.walls
//faces	worldedit.region.faces
//smooth	worldedit.region.smooth
//snowsmooth	worldedit.region.snowsmooth
//move	worldedit.region.move
//stack	worldedit.region.stack
//regen	worldedit.regen
//deform	worldedit.region.deform
//hollow	worldedit.region.hollow
//forest	worldedit.region.forest
//flora	worldedit.region.flora
//update	worldedit.update
//hcyl	worldedit.generation.cylinder
//cyl	worldedit.generation.cylinder
//cone	worldedit.generation.cone
//hsphere	worldedit.generation.sphere
//sphere	worldedit.generation.sphere
/forestgen	worldedit.generation.forest
/pumpkins	worldedit.generation.pumpkins
//feature	worldedit.generation.feature
//structure	worldedit.generation.structure
//hpyramid	worldedit.generation.pyramid
//pyramid	worldedit.generation.pyramid
//generate	worldedit.generation.shape
//generatebiome	worldedit.generation.shape.biome
/schematic	worldedit.schematic.delete, worldedit.schematic.list, worldedit.clipboard.load, worldedit.schematic.save, worldedit.schematic.formats, worldedit.schematic.load, worldedit.clipboard.save, worldedit.clipboard.share, worldedit.schematic.share
/schematic delete	worldedit.schematic.delete
/schematic formats	worldedit.schematic.formats
/schematic list	worldedit.schematic.list

continues on next page

Table 2 – continued from previous page

Command	Permission
/schematic load	worldedit.clipboard.load, worldedit.schematic.load
/schematic save	worldedit.clipboard.save, worldedit.schematic.save
/schematic share	worldedit.clipboard.share, worldedit.schematic.share
//copy	worldedit.clipboard.copy
//cut	worldedit.clipboard.cut
//paste	worldedit.clipboard.paste
//rotate	worldedit.clipboard.rotate
//flip	worldedit.clipboard.flip
/clearclipboard	worldedit.clipboard.clear
//revolve	worldedit.revolve
/tool	
/tool cycler	worldedit.tool.data-cycler
/tool deltree	worldedit.tool.deltree
/tool farwand	worldedit.tool.farwand
/tool floodfill	worldedit.tool.flood-fill
/tool info	worldedit.tool.info
/tool lrbuild	worldedit.tool.lrbuild
/tool navwand	worldedit.setwand
/tool none	
/tool repl	worldedit.tool.replacer
/tool selwand	worldedit.setwand
/tool stacker	worldedit.tool.stack
/tool tree	worldedit.tool.tree
/none	
/selwand	worldedit.setwand
/navwand	worldedit.setwand
/info	worldedit.tool.info
/tree	worldedit.tool.tree
/repl	worldedit.tool.replacer
/cycler	worldedit.tool.data-cycler
/floodfill	worldedit.tool.flood-fill
/deltree	worldedit.tool.deltree
/farwand	worldedit.tool.farwand
/lrbuild	worldedit.tool.lrbuild
//	worldedit.superpickaxe
/mask	worldedit.brush.options.mask
/material	worldedit.brush.options.material
/range	worldedit.brush.options.range
/size	worldedit.brush.options.size
/tracemask	worldedit.brush.options.tracemask
/superpickaxe	worldedit.superpickaxe.area, worldedit.superpickaxe.recursive, worldedit.superpickaxe
/superpickaxe area	worldedit.superpickaxe.area
/superpickaxe recursive	worldedit.superpickaxe.recursive
/superpickaxe single	worldedit.superpickaxe
/brush	
/brush apply	worldedit.brush.apply
/brush apply forest	
/brush apply item	worldedit.brush.item
/brush apply set	

continues on next page

Table 2 – continued from previous page

Command	Permission
/brush biome	worldedit.brush.biome
/brush butcher	worldedit.brush.butcher
/brush clipboard	worldedit.brush.clipboard
/brush cylinder	worldedit.brush.cylinder
/brush deform	worldedit.brush.deform
/brush dilate	worldedit.brush.morph
/brush erode	worldedit.brush.morph
/brush extinguish	worldedit.brush.ex
/brush feature	worldedit.brush.feature
/brush forest	worldedit.brush.forest
/brush gravity	worldedit.brush.gravity
/brush heightmap	worldedit.brush.heightmap
/brush lower	worldedit.brush.lower
/brush morph	worldedit.brush.morph
/brush none	
/brush paint	worldedit.brush.paint
/brush paint forest	
/brush paint item	worldedit.brush.item
/brush paint set	
/brush raise	worldedit.brush.raise
/brush set	worldedit.brush.set
/brush smooth	worldedit.brush.smooth
/brush snow	worldedit.brush.snow
/brush snowsmooth	worldedit.brush.snowsmooth
/brush sphere	worldedit.brush.sphere
/brush splatter	worldedit.brush.splatter
/biomelist	worldedit.biome.list
/biomeinfo	worldedit.biome.info
//setbiome	worldedit.biome.set
//replacebiome	worldedit.biome.set
/chunkinfo	worldedit.chunkinfo
/listchunks	worldedit.listchunks
/delchunks	worldedit.delchunks
/restore	worldedit.snapshots.restore
/snapshot	worldedit.snapshots.restore, worldedit.snapshots.list
/snapshot after	worldedit.snapshots.restore
/snapshot before	worldedit.snapshots.restore
/snapshot list	worldedit.snapshots.list
/snapshot sel	worldedit.snapshots.restore
/snapshot use	worldedit.snapshots.restore
/cs	worldedit.scripting.execute
/.s	worldedit.scripting.execute
//fill	worldedit.fill
//fillr	worldedit.fill.recursive
//drain	worldedit.drain
/fixlava	worldedit.fixlava
/fixwater	worldedit.fixwater
/removeabove	worldedit.removeabove
/removebelow	worldedit.removebelow
/removenear	worldedit.removenear

continues on next page

Table 2 – continued from previous page

Command	Permission
/replacenear	worldedit.replacenear
/snow	worldedit.snow
/thaw	worldedit.thaw
/green	worldedit.green
/extinguish	worldedit.extinguish
/butcher	worldedit.butcher
/remove	worldedit.remove
//calculate	worldedit.calc
//help	worldedit.help

1.4.2 Other Permissions

Permission	Explanation
worldedit.navigation.jumpto.tool	Allows usage of the navigation wand's /jumpto shortcut (left click).
worldedit.navigation.thru.tool	Allows usage of the navigation wand's /thru shortcut (right click).
worldedit.anyblock	Allows usage of blocks in the *disallowed-blocks* config option.
worldedit.limit.unrestricted	Allows setting the limit via the //limit *command* higher than the maximum in the *configuration*, as well as other limit bypasses.
worldedit.timeout.unrestricted	Allows setting the calculation timeout via the //timeout *command* higher than the maximum in the *configuration*.
worldedit.inventory.unrestricted	Override the use-inventory option if enabled in the *configuration*.
worldedit.override.bedrock	Allows breaking of bedrock with the super-pickaxe tool.
worldedit.override.data-cycler	Allows cycling non-whitelisted blocks with the data cycler tool.
worldedit.setnbt	Allows setting extra data on blocks (such as signs, chests, etc).
worldedit.report.pastebin	Allows uploading report files to pastebin automatically for the /worldedit report *command*.
worldedit.scripting.execute.<filename>	Allows using the CraftScript with the given filename.

1.5 Commands

Note

Arguments enclosed in `[ ]` are optional, those enclosed in `< >` are required.

Tip

You can access a command listing in-game via the `//help` command.

1.5.1 General Commands

`/worldedit` (or `/we`)

Description	WorldEdit commands
Usage	`/worldedit <help|version|trace|reload|cui|tz|report>`

`/worldedit help`

Description	Displays help for WorldEdit commands
Permissions	`worldedit.help`
Usage	`/worldedit help [-s] [-p <page>] [command...]`
`[-s]`	List sub-commands of the given command, if applicable
`[-p <page>]`	The page to retrieve
`[command...]`	The command to retrieve help for

`/worldedit version` (or `/worldedit ver`)

Description	Get WorldEdit version
Usage	`/worldedit version`

`/worldedit trace`

Description	Toggles trace hook
Usage	`/worldedit trace [hookMode]`
`[hookMode]`	The mode to set the trace hook to

`/worldedit reload`

Description	Reload configuration
Permissions	`worldedit.reload`
Usage	`/worldedit reload`

`/worldedit cui`

Description	Complete CUI handshake (internal usage)
Usage	`/worldedit cui`

`/worldedit tz`

Description	Set your timezone for snapshots
Usage	`/worldedit tz <timezone>`
`<timezone>`	The timezone to set

`/worldedit report`

Description	Writes a report on WorldEdit
Permissions	`worldedit.report`
Usage	`/worldedit report [-p]`
`[-p]`	Pastebins the report

`/undo` (or `//undo`)

Description	Undoes the last action (from history)
Permissions	`worldedit.history.undo`, `worldedit.history.undo.self`
Usage	`/undo [times] [player]`
`[times]`	Number of undoes to perform
`[player]`	Undo this player's operations

`/redo` (or `//redo`)

Description	Redoes the last action (from history)
Permissions	`worldedit.history.redo`, `worldedit.history.redo.self`
Usage	`/redo [times] [player]`
`[times]`	Number of redoes to perform
`[player]`	Redo this player's operations

`/clearhistory` (or `//clearhistory`)

Description	Clear your history
Permissions	`worldedit.history.clear`
Usage	`/clearhistory`

`//limit`

Description	Modify block change limit
Permissions	`worldedit.limit`
Usage	`//limit [limit]`
`[limit]`	The limit to set

`//timeout`

Description	Modify evaluation timeout time.
Permissions	`worldedit.timeout`
Usage	`//timeout [limit]`
`[limit]`	The timeout time to set

`//fast`

> **Warning**
>
> This command is deprecated. //fast duplicates //perf and will be removed in WorldEdit 8.
> Please use `//perf` instead.

Description	Toggle fast mode
Permissions	`worldedit.fast`
Usage	`//fast [fastMode]`
`[fastMode]`	The new fast mode state

`//perf`

Description	Toggle side effects for performance Note that this command is GOING to change in the future. Do not depend on the exact format of this command yet.
Permissions	`worldedit.perf`
Usage	`//perf [-h] [sideEffect] [newState]`
`[sideEffect]`	The side effect
`[newState]`	The new side effect state
`[-h]`	Show the info box

`//reorder`

Description	Sets the reorder mode of WorldEdit
Permissions	`worldedit.reorder`
Usage	`//reorder [reorderMode]`
`[reorderMode]`	The reorder mode

`//drawsel`

Description	Toggle drawing the current selection
Permissions	`worldedit.drawsel`
Usage	`//drawsel [drawSelection]`
`[drawSelection]`	The new draw selection state

`//world`

Description	Sets the world override
Permissions	`worldedit.world`
Usage	`//world [world]`
`[world]`	The world override

`//watchdog`

Description	Changes watchdog hook state. This is dependent on platform implementation. Not all platforms support watchdog hooks, or contain a watchdog.
Permissions	`worldedit.watchdog`
Usage	`//watchdog [hookMode]`
`[hookMode]`	The mode to set the watchdog hook to

`/gmask` (or `//gmask`)

Description	Set the global mask
Permissions	`worldedit.global-mask`
Usage	`/gmask [mask]`
`[mask]`	The mask to set

`/toggleplace` (or `//toggleplace`)

Description	Switch between your position and pos1 for placement
Usage	`/toggleplace`

`/placement` (or `//placement`)

Description	Select which placement to use
Permissions	`worldedit.placement`
Usage	`/placement <placementType> [multiplier] [offset]`
`<placementType>`	Which placement type to use
`[multiplier]`	number of times to apply the offset
`[offset]`	How much to offset from it placement to use

`/searchitem` (or `//searchitem`, `//l`, `//search`)

Description	Search for an item
Permissions	`worldedit.searchitem`
Usage	`/searchitem [-bi] [-p <page>] <query...>`
`[-b]`	Only search for blocks
`[-i]`	Only search for items
`[-p <page>]`	Page of results to return
`<query...>`	Search query

`//registry`

Description	Search through the given registry
Permissions	`worldedit.registry`
Usage	`//registry <registry> [-p <page>] [queryBits...]`
`<registry>`	The registry to search through
`[-p <page>]`	Page of results to return
`[queryBits...]`	Search query

1.5.2 Navigation Commands

/unstuck (or /!)

Description	Escape from being stuck inside a block
Permissions	worldedit.navigation.unstuck
Usage	/unstuck

/ascend (or /asc)

Description	Go up a floor
Permissions	worldedit.navigation.ascend
Usage	/ascend [levels]
[levels]	# of levels to ascend

/descend (or /desc)

Description	Go down a floor
Permissions	worldedit.navigation.descend
Usage	/descend [levels]
[levels]	# of levels to descend

/ceil

Description	Go to the ceiling
Permissions	worldedit.navigation.ceiling
Usage	/ceil [-fg] [clearance]
[clearance]	# of blocks to leave above you
[-f]	Force using flight to keep you still
[-g]	Force using glass to keep you still

/thru

Description	Pass through walls
Permissions	worldedit.navigation.thru.command
Usage	/thru

/jumpto (or /j)

Description	Teleport to a location
Permissions	worldedit.navigation.jumpto.command
Usage	/jumpto

/up

Description	Go upwards some distance
Permissions	`worldedit.navigation.up`
Usage	`/up [-fg] <distance>`
`<distance>`	Distance to go upwards
`[-f]`	Force using flight to keep you still
`[-g]`	Force using glass to keep you still

1.5.3 Selection Commands

//pos

Description	Set positions
Permissions	`worldedit.selection.pos`
Usage	`//pos [pos1] [pos2...] [-s <selectorChoice>]`
`[pos1]`	Coordinates to set the primary position to. Defaults to the player position if not passed.
`[pos2...]`	Coordinates to add as secondary positions. Defaults to the player position if not passed.
`[-s <selectorChoice>]`	Selector to switch to

//pos1

Description	Set position 1
Permissions	`worldedit.selection.pos`
Usage	`//pos1 [coordinates]`
`[coordinates]`	Coordinates to set position 1 to

//pos2

Description	Set position 2
Permissions	`worldedit.selection.pos`
Usage	`//pos2 [coordinates]`
`[coordinates]`	Coordinates to set position 2 to

//hpos1

Description	Set position 1 to targeted block
Permissions	`worldedit.selection.hpos`
Usage	`//hpos1`

//hpos2

Description	Set position 2 to targeted block
Permissions	`worldedit.selection.hpos`
Usage	`//hpos2`

//chunk

Description	Set the selection to your current chunk. This command selects 256-block-tall areas, which can be specified by the y-coordinate. E.g. -c x,1,z will select from y=256 to y=511.
Permissions	`worldedit.selection.chunk`
Usage	`//chunk [-cs] [coordinates]`
`[coordinates]`	The chunk to select
`[-s]`	Expand your selection to encompass all chunks that are part of it
`[-c]`	Use chunk coordinates instead of block coordinates

//wand

Description	Get the wand item You must have also have permission to use at least one of the features of the requested wand.
Permissions	`worldedit.wand`
Usage	`//wand [-n]`
`[-n]`	Get a navigation wand

/toggleeditwand

Description	Remind the user that the wand is now a tool and can be unbound with /tool none.
Permissions	`worldedit.wand.toggle`
Usage	`/toggleeditwand`

//contract

Description	Contract the selection area
Permissions	`worldedit.selection.contract`
Usage	`//contract <amount> [reverseAmount] [direction]`
`<amount>`	Amount to contract the selection by
`[reverseAmount]`	Amount to contract the selection by in the other direction
`[direction]`	Direction to contract

//shift

Description	Shift the selection area
Permissions	`worldedit.selection.shift`
Usage	`//shift <amount> [direction]`
`<amount>`	Amount to shift the selection by
`[direction]`	Direction to contract

//outset

Description	Outset the selection area
Permissions	`worldedit.selection.outset`
Usage	`//outset [-hv] <amount>`
`<amount>`	Amount to expand the selection by in all directions
`[-h]`	Only expand horizontally
`[-v]`	Only expand vertically

//inset

Description	Inset the selection area
Permissions	`worldedit.selection.inset`
Usage	`//inset [-hv] <amount>`
`<amount>`	Amount to contract the selection by in all directions
`[-h]`	Only contract horizontally
`[-v]`	Only contract vertically

//trim

Description	Minimize the selection to encompass matching blocks
Permissions	`worldedit.selection.trim`
Usage	`//trim [mask]`
`[mask]`	Mask of blocks to keep within the selection

//size

Description	Get information about the selection
Permissions	`worldedit.selection.size`
Usage	`//size [-c]`
`[-c]`	Get clipboard info instead

//count

Description	Counts the number of blocks matching a mask
Permissions	`worldedit.analysis.count`
Usage	`//count <mask>`
`<mask>`	The mask of blocks to match

`//distr`

Description	Get the distribution of blocks in the selection
Permissions	`worldedit.analysis.distr`
Usage	`//distr [-cd] [-p <page>] [-m <sourceMask>]`
`[-c]`	Get the distribution of the clipboard instead
`[-d]`	Separate blocks by state
`[-p <page>]`	Gets page from a previous distribution.
`[-m <sourceMask>]`	Only include blocks matching the given mask

`//sel` (or `/;`, `//desel`, `//deselect`)

Description	Choose a region selector
Usage	`//sel [-d] [selectorChoiceOrList]`
`[selectorChoiceOrList]`	Selector to switch to
`[-d]`	Set default selector

`//expand`

Description	Expand the selection area
Permissions	`worldedit.selection.expand`
Usage	`//expand <vert|<amount> [reverseAmount] [direction]>`
`<amount>`	Amount to expand the selection by, can be *vert* to expand to the whole vertical column
`[reverseAmount]`	Amount to expand the selection by in the other direction
`[direction]`	Direction to expand

`//expand vert`

Description	Vertically expand the selection to world limits.
Usage	`//expand vert`

1.5.4 Region Commands

`//set`

Description	Sets all the blocks in the region
Permissions	`worldedit.region.set`
Usage	`//set <pattern>`
`<pattern>`	The pattern of blocks to set

`//line`

Description	Draws line segments between cuboid selection corners or convex polyhedral selection vertices Can only be used with a cuboid selection or a convex polyhedral selection
Permissions	`worldedit.region.line`
Usage	`//line [-h] <pattern> [thickness]`
`<pattern>`	The pattern of blocks to place
`[thickness]`	The thickness of the line
`[-h]`	Generate only a shell

`//curve`

Description	Draws a spline through selected points Can only be used with a convex polyhedral selection
Permissions	`worldedit.region.curve`
Usage	`//curve [-h] <pattern> [thickness]`
`<pattern>`	The pattern of blocks to place
`[thickness]`	The thickness of the curve
`[-h]`	Generate only a shell

`//replace` (or `//re`, `//rep`)

Description	Replace all blocks in the selection with another
Permissions	`worldedit.region.replace`
Usage	`//replace [from] <to>`
`[from]`	The mask representing blocks to replace
`<to>`	The pattern of blocks to replace with

`//overlay`

Description	Set a block on top of blocks in the region
Permissions	`worldedit.region.overlay`
Usage	`//overlay <pattern>`
`<pattern>`	The pattern of blocks to overlay

//center (or //middle)

Description	Set the center block(s)
Permissions	worldedit.region.center
Usage	//center <pattern>
<pattern>	The pattern of blocks to set

//naturalize

Description	3 layers of dirt on top then rock below
Permissions	worldedit.region.naturalize
Usage	//naturalize

//walls

Description	Build the four sides of the selection
Permissions	worldedit.region.walls
Usage	//walls <pattern>
<pattern>	The pattern of blocks to set

//faces (or //outline)

Description	Build the walls, ceiling, and floor of a selection
Permissions	worldedit.region.faces
Usage	//faces <pattern>
<pattern>	The pattern of blocks to set

//smooth

Description	Smooth the elevation in the selection Example: '//smooth 1 grass_block,dirt,stone' would only smooth natural surface terrain.
Permissions	worldedit.region.smooth
Usage	//smooth [iterations] [mask]
[iterations]	# of iterations to perform
[mask]	The mask of blocks to use as the height map

//snowsmooth

Description	Smooth the elevation in the selection with snow layers Example: '//snowsmooth 1 -m snow_block,snow' would only smooth snow terrain.
Permissions	`worldedit.region.snowsmooth`
Usage	`//snowsmooth [iterations] [-l <snowBlockCount>] [-m <mask>]`
`[iterations]`	# of iterations to perform
`[-l <snowBlockCount>]`	Set the amount of snow blocks under the snow
`[-m <mask>]`	The mask of blocks to use as the height map

//move

Description	Move the contents of the selection
Permissions	`worldedit.region.move`
Usage	`//move [-abes] [multiplier] [offset] [replace] [-m <mask>]`
`[multiplier]`	number of times to apply the offset
`[offset]`	The offset to move
`[replace]`	The pattern of blocks to leave
`[-s]`	Shift the selection to the target location
`[-a]`	Ignore air blocks
`[-e]`	Also copy entities
`[-b]`	Also copy biomes
`[-m <mask>]`	Set the include mask, non-matching blocks become air

//stack

Description	Repeat the contents of the selection
Permissions	`worldedit.region.stack`
Usage	`//stack [-abers] [count] [offset] [-m <mask>]`
`[count]`	# of copies to stack
`[offset]`	How far to move the contents each stack
`[-s]`	Shift the selection to the last stacked copy
`[-a]`	Ignore air blocks
`[-e]`	Also copy entities
`[-b]`	Also copy biomes
`[-r]`	Use block units
`[-m <mask>]`	Set the include mask, non-matching blocks become air

//regen

Description	Regenerates the contents of the selection
Permissions	`worldedit.regen`
Usage	`//regen [-bc] [seed]`
`[seed]`	The seed to regenerate with, otherwise uses world seed
`[-b]`	Regenerate biomes as well
`[-c]`	Regenerate to the clipboard

`//deform`

Description	Deforms a selected region with an expression The expression is executed for each block and is expected to modify the variables x, y and z to point to a new block to fetch. For details, see https://ehub.to/we/expr
Permissions	`worldedit.region.deform`
Usage	`//deform [-clor] <expression...>`
`<expression...>`	The expression to use
`[-r]`	Use the game's coordinate origin
`[-o]`	Use the placement's coordinate origin
`[-c]`	Use the selection's center as origin
`[-l]`	Fetch from the clipboard instead of the world

`//hollow`

Description	Hollows out the object contained in this selection Thickness is measured in manhattan distance.
Permissions	`worldedit.region.hollow`
Usage	`//hollow [thickness] [pattern]`
`[thickness]`	Thickness of the shell to leave
`[pattern]`	The pattern of blocks to replace the hollowed area with

`//forest`

Description	Make a forest within the region
Permissions	`worldedit.region.forest`
Usage	`//forest [type] [density]`
`[type]`	The type of tree to place
`[density]`	The density of the forest

`//flora`

Description	Make flora within the region
Permissions	`worldedit.region.flora`
Usage	`//flora [density]`
`[density]`	The density of the forest

`//update`

Description	Apply side effects to your selection
Permissions	`worldedit.update`
Usage	`//update [sideEffectSet]`
`[sideEffectSet]`	The side effects

1.5.5 Generation Commands

`//hcyl`

Description	Generates a hollow cylinder.
Permissions	`worldedit.generation.cylinder`
Usage	`//hcyl <pattern> <radii> [height]`
`<pattern>`	The pattern of blocks to generate
`<radii>`	The radii of the cylinder. 1st is N/S, 2nd is E/W
`[height]`	The height of the cylinder

`//cyl`

Description	Generates a cylinder.
Permissions	`worldedit.generation.cylinder`
Usage	`//cyl [-h] <pattern> <radii> [height]`
`<pattern>`	The pattern of blocks to generate
`<radii>`	The radii of the cylinder. 1st is N/S, 2nd is E/W
`[height]`	The height of the cylinder
`[-h]`	Make a hollow cylinder

`//cone`

Description	Generates a cone.
Permissions	`worldedit.generation.cone`
Usage	`//cone [-h] <pattern> <radii> [height] [thickness]`
`<pattern>`	The pattern of blocks to generate
`<radii>`	The radii of the cone. 1st is N/S, 2nd is E/W
`[height]`	The height of the cone
`[-h]`	Make a hollow cone
`[thickness]`	Thickness of the hollow cone

//hsphere

Description	Generates a hollow sphere.
Permissions	`worldedit.generation.sphere`
Usage	`//hsphere [-r] <pattern> <radii>`
`<pattern>`	The pattern of blocks to generate
`<radii>`	The radii of the sphere. Order is N/S, U/D, E/W
`[-r]`	Raise the bottom of the sphere to the placement position

//sphere

Description	Generates a filled sphere.
Permissions	`worldedit.generation.sphere`
Usage	`//sphere [-hr] <pattern> <radii>`
`<pattern>`	The pattern of blocks to generate
`<radii>`	The radii of the sphere. Order is N/S, U/D, E/W
`[-r]`	Raise the bottom of the sphere to the placement position
`[-h]`	Make a hollow sphere

/forestgen

Description	Generate a forest
Permissions	`worldedit.generation.forest`
Usage	`/forestgen [size] [type] [density]`
`[size]`	The size of the forest, in blocks
`[type]`	The type of forest
`[density]`	The density of the forest, between 0 and 100

/pumpkins

Description	Generate pumpkin patches
Permissions	`worldedit.generation.pumpkins`
Usage	`/pumpkins [size]`
`[size]`	The size of the patch

//feature

Description	Generate Minecraft features
Permissions	`worldedit.generation.feature`
Usage	`//feature <feature>`
`<feature>`	The feature

//structure

Description	Generate Minecraft structures
Permissions	worldedit.generation.structure
Usage	//structure <feature>
<feature>	The structure

//hpyramid

Description	Generate a hollow pyramid
Permissions	worldedit.generation.pyramid
Usage	//hpyramid <pattern> <size>
<pattern>	The pattern of blocks to set
<size>	The size of the pyramid

//pyramid

Description	Generate a filled pyramid
Permissions	worldedit.generation.pyramid
Usage	//pyramid [-h] <pattern> <size>
<pattern>	The pattern of blocks to set
<size>	The size of the pyramid
[-h]	Make a hollow pyramid

//generate (or //gen, //g)

Description	Generates a shape according to a formula. For details, see https://ehub.to/we/expr
Permissions	worldedit.generation.shape
Usage	//generate [-chor] <pattern> <expression...>
<pattern>	The pattern of blocks to set
<expression...>	Expression to test block placement locations and set block type
[-h]	Generate a hollow shape
[-r]	Use the game's coordinate origin
[-o]	Use the placement's coordinate origin
[-c]	Use the selection's center as origin

//generatebiome (or //genbiome, //gb)

Description	Sets biome according to a formula. For details, see https://ehub.to/we/expr
Permissions	`worldedit.generation.shape.biome`
Usage	`//generatebiome [-chor] <target> <expression...>`
`<target>`	The biome type to set
`<expression...>`	Expression to test block placement locations and set biome type
`[-h]`	Generate a hollow shape
`[-r]`	Use the game's coordinate origin
`[-o]`	Use the placement's coordinate origin
`[-c]`	Use the selection's center as origin

1.5.6 Schematic and Clipboard Commands

`/schematic` (or `/schem`, `//schematic`, `//schem`)

Description	Schematic commands for saving/loading areas
Permissions	`worldedit.schematic.delete`, `worldedit.schematic.list`, `worldedit.clipboard.load`, `worldedit.schematic.save`, `worldedit.schematic.formats`, `worldedit.schematic.load`, `worldedit.clipboard.save`, `worldedit.clipboard.share`, `worldedit.schematic.share`
Usage	`/schematic <list|formats|load|delete|save|share>`

`/schematic list` (or `/schematic all`, `/schematic ls`)

Description	List saved schematics Note: Format is not fully verified until loading.
Permissions	`worldedit.schematic.list`
Usage	`/schematic list [-dn] [-p <page>]`
`[-p <page>]`	Page to view.
`[-d]`	Sort by date, oldest first
`[-n]`	Sort by date, newest first

`/schematic formats` (or `/schematic listformats`, `/schematic f`)

Description	List available formats
Permissions	`worldedit.schematic.formats`
Usage	`/schematic formats`

`/schematic load`

Description	Load a schematic into your clipboard
Permissions	`worldedit.clipboard.load`, `worldedit.schematic.load`
Usage	`/schematic load <schematic> [format]`
`<schematic>`	File name.
`[format]`	Format name.

`/schematic delete` (or `/schematic d`)

Description	Delete a saved schematic
Permissions	`worldedit.schematic.delete`
Usage	`/schematic delete <schematic>`
`<schematic>`	File name.

`/schematic save`

Description	Save your clipboard into a schematic file
Permissions	`worldedit.clipboard.save`, `worldedit.schematic.save`
Usage	`/schematic save [-f] <filename> [format]`
`<filename>`	File name.
`[format]`	Format name.
`[-f]`	Overwrite an existing file.

`/schematic share`

Description	Share your clipboard as a schematic online
Permissions	`worldedit.clipboard.share`, `worldedit.schematic.share`
Usage	`/schematic share [schematicName] [destination] [format]`
`[schematicName]`	Schematic name. Defaults to name-millis
`[destination]`	Share location
`[format]`	Format name

`//copy`

Description	Copy the selection to the clipboard
Permissions	`worldedit.clipboard.copy`
Usage	`//copy [-be] [-m <mask>]`
`[-e]`	Also copy entities
`[-b]`	Also copy biomes
`[-m <mask>]`	Set the include mask, non-matching blocks become air

//cut

Description	Cut the selection to the clipboard
Permissions	worldedit.clipboard.cut
Usage	//cut [-be] [leavePattern] [-m <mask>]
[leavePattern]	Pattern to leave in place of the selection
[-e]	Also cut entities
[-b]	Also copy biomes, source biomes are unaffected
[-m <mask>]	Set the exclude mask, non-matching blocks become air

//paste

Description	Paste the clipboard's contents
Permissions	worldedit.clipboard.paste
Usage	//paste [-abenosv] [-m <sourceMask>]
[-a]	Skip air blocks
[-v]	Include structure void blocks
[-o]	Paste at the original position
[-s]	Select the region after pasting
[-n]	No paste, select only. (Implies -s)
[-e]	Paste entities if available
[-b]	Paste biomes if available
[-m <sourceMask>]	Only paste blocks matching this mask

//rotate

Description	Rotate the contents of the clipboard Non-destructively rotate the contents of the clipboard. Angles are provided in degrees and a positive angle will result in a clockwise rotation. Multiple rotations can be stacked. Interpolation is not performed so angles should be a multiple of 90 degrees.
Permissions	worldedit.clipboard.rotate
Usage	//rotate <rotateY> [rotateX] [rotateZ]
<rotateY>	Amount to rotate on the y-axis
[rotateX]	Amount to rotate on the x-axis
[rotateZ]	Amount to rotate on the z-axis

//flip (or //mirror)

Description	Flip the contents of the clipboard across the origin
Permissions	worldedit.clipboard.flip
Usage	//flip [direction]
[direction]	The direction to flip, defaults to look direction.

/clearclipboard

Description	Clear your clipboard
Permissions	`worldedit.clipboard.clear`
Usage	`/clearclipboard`

//revolve

Description	Revolve the selection around a vertical axis
Permissions	`worldedit.revolve`
Usage	`//revolve [-ber] <pasteCount> [-m <mask>]`
`<pasteCount>`	The number of pastes
`[-m <mask>]`	Set the source mask, non-matching blocks are not revolved
`[-r]`	Perform revolutions in reverse (counter-clockwise)
`[-e]`	Copy entities
`[-b]`	Copy biomes

1.5.7 Tool Commands

/tool

Description	Binds a tool to the item in your hand
Usage	`/tool <cycler|repl|farwand|floodfill|deltree|stacker|selwand|tree|none|`

/tool cycler

Description	Block data cycler tool
Permissions	`worldedit.tool.data-cycler`
Usage	`/tool cycler`

/tool repl

Description	Block replacer tool
Permissions	`worldedit.tool.replacer`
Usage	`/tool repl <pattern>`
`<pattern>`	The pattern of blocks to place

/tool farwand

Description	Wand at a distance tool
Permissions	`worldedit.tool.farwand`
Usage	`/tool farwand`

`/tool floodfill` (or `/tool flood`)

Description	Flood fill tool
Permissions	`worldedit.tool.flood-fill`
Usage	`/tool floodfill <pattern> <range>`
`<pattern>`	The pattern to flood fill
`<range>`	The range to perform the fill

`/tool deltree`

Description	Floating tree remover tool
Permissions	`worldedit.tool.deltree`
Usage	`/tool deltree`

`/tool stacker`

Description	Block stacker tool
Permissions	`worldedit.tool.stack`
Usage	`/tool stacker [range] [mask]`
`[range]`	The max range of the stack
`[mask]`	The mask to stack until

`/tool selwand`

Description	Selection wand tool
Permissions	`worldedit.setwand`
Usage	`/tool selwand`

`/tool tree`

Description	Tree generator tool
Permissions	`worldedit.tool.tree`
Usage	`/tool tree [type]`
`[type]`	Type of tree to generate

`/tool none` (or `/tool unbind`)

Description	Unbind a bound tool from your current item
Usage	`/tool none`

`/tool info`

Description	Block information tool
Permissions	`worldedit.tool.info`
Usage	`/tool info`

`/tool lrbuild`

Description	Long-range building tool
Permissions	`worldedit.tool.lrbuild`
Usage	`/tool lrbuild <primary> <secondary>`
`<primary>`	Pattern to set on left-click
`<secondary>`	Pattern to set on right-click

`/tool navwand`

Description	Navigation wand tool
Permissions	`worldedit.setwand`
Usage	`/tool navwand`

`/none`

Warning

This command is deprecated. Global tool names cause conflicts and will be removed in WorldEdit 8.
Please use `/tool none` instead.

Description	Unbind a bound tool from your current item
Usage	`/none`

`/selwand` (or `//selwand`)

Warning

This command is deprecated. Global tool names cause conflicts and will be removed in WorldEdit 8.
Please use `/tool selwand` instead.

Description	Selection wand tool
Permissions	`worldedit.setwand`
Usage	`/selwand`

`/navwand` (or `//navwand`)

Warning

This command is deprecated. Global tool names cause conflicts and will be removed in WorldEdit 8.
Please use `/tool navwand` instead.

Description	Navigation wand tool
Permissions	`worldedit.setwand`
Usage	`/navwand`

`/info`

Warning

This command is deprecated. Global tool names cause conflicts and will be removed in WorldEdit 8.
Please use `/tool info` instead.

Description	Block information tool
Permissions	`worldedit.tool.info`
Usage	`/info`

`/tree`

Warning

This command is deprecated. Global tool names cause conflicts and will be removed in WorldEdit 8.
Please use `/tool tree` instead.

Description	Tree generator tool
Permissions	`worldedit.tool.tree`
Usage	`/tree [type]`
`[type]`	Type of tree to generate

`/repl`

Warning

This command is deprecated. Global tool names cause conflicts and will be removed in WorldEdit 8.
Please use `/tool repl` instead.

Description	Block replacer tool
Permissions	`worldedit.tool.replacer`
Usage	`/repl <pattern>`
`<pattern>`	The pattern of blocks to place

`/cycler`

Warning

This command is deprecated. Global tool names cause conflicts and will be removed in WorldEdit 8.
Please use `/tool cycler` instead.

Description	Block data cycler tool
Permissions	`worldedit.tool.data-cycler`
Usage	`/cycler`

`/floodfill` (or `/flood`)

Warning

This command is deprecated. Global tool names cause conflicts and will be removed in WorldEdit 8.
Please use `/tool floodfill` instead.

Description	Flood fill tool
Permissions	`worldedit.tool.flood-fill`
Usage	`/floodfill <pattern> <range>`
`<pattern>`	The pattern to flood fill
`<range>`	The range to perform the fill

`/deltree`

Warning

This command is deprecated. Global tool names cause conflicts and will be removed in WorldEdit 8.
Please use `/tool deltree` instead.

Description	Floating tree remover tool
Permissions	`worldedit.tool.deltree`
Usage	`/deltree`

`/farwand`

Warning

This command is deprecated. Global tool names cause conflicts and will be removed in WorldEdit 8.
Please use `/tool farwand` instead.

Description	Wand at a distance tool
Permissions	`worldedit.tool.farwand`
Usage	`/farwand`

`/lrbuild` (or `//lrbuild`)

Warning

This command is deprecated. Global tool names cause conflicts and will be removed in WorldEdit 8.
Please use `/tool lrbuild` instead.

Description	Long-range building tool
Permissions	`worldedit.tool.lrbuild`
Usage	`/lrbuild <primary> <secondary>`
`<primary>`	Pattern to set on left-click
`<secondary>`	Pattern to set on right-click

`//` (or `/,`)

Description	Toggle the super pickaxe function
Permissions	`worldedit.superpickaxe`
Usage	`// [superPickaxe]`
`[superPickaxe]`	The new super pickaxe state

`/mask`

Description	Set the brush mask
Permissions	`worldedit.brush.options.mask`
Usage	`/mask [mask]`
`[mask]`	The mask to set

`/material` (or `//material`)

Description	Set the brush material
Permissions	`worldedit.brush.options.material`
Usage	`/material <pattern>`
`<pattern>`	The pattern of blocks to use

`/range`

Description	Set the brush range
Permissions	`worldedit.brush.options.range`
Usage	`/range <range>`
`<range>`	The range of the brush

`/size`

Description	Set the brush size
Permissions	`worldedit.brush.options.size`
Usage	`/size <size>`
`<size>`	The size of the brush

`/tracemask`

Description	Set the mask used to stop tool traces
Permissions	`worldedit.brush.options.tracemask`
Usage	`/tracemask [mask]`
`[mask]`	The trace mask to set

1.5.8 Super Pickaxe Commands

`/superpickaxe` (or **`/pickaxe`**, **`/sp`**)

Description	Super-pickaxe commands
Permissions	`worldedit.superpickaxe.area`, `worldedit.superpickaxe.recursive`, `worldedit.superpickaxe`
Usage	`/superpickaxe <single|area|recursive>`

`/superpickaxe single`

Description	Enable the single block super pickaxe mode
Permissions	`worldedit.superpickaxe`
Usage	`/superpickaxe single`

`/superpickaxe area`

Description	Enable the area super pickaxe pickaxe mode
Permissions	`worldedit.superpickaxe.area`
Usage	`/superpickaxe area <range>`
`<range>`	The range of the area pickaxe

`/superpickaxe recursive` (or **`/superpickaxe recur`**)

Description	Enable the recursive super pickaxe pickaxe mode
Permissions	`worldedit.superpickaxe.recursive`
Usage	`/superpickaxe recursive <range>`
`<range>`	The range of the recursive pickaxe

1.5.9 Brush Commands

/brush (or **/br**, **//brush**, **//br**)

Description	Brushing commands
Usage	`/brush <forest|splatter|butcher|paint|none|clipboard|snowsmooth|gravity`

/brush forest

Description	Forest brush, creates a forest in the area
Permissions	`worldedit.brush.forest`
Usage	`/brush forest <shape> [radius] [density] <type>`
`<shape>`	The shape of the region
`[radius]`	The size of the brush
`[density]`	The density of the brush
`<type>`	The type of tree to use

/brush splatter (or **/brush splat**)

Description	Choose the splatter brush
Permissions	`worldedit.brush.splatter`
Usage	`/brush splatter <pattern> [radius] [decay]`
`<pattern>`	The pattern of blocks to set
`[radius]`	The radius of the splatter
`[decay]`	The decay of the splatter between 0 and 10

/brush butcher (or **/brush kill**)

Description	Butcher brush, kills mobs within a radius
Permissions	`worldedit.brush.butcher`
Usage	`/brush butcher [-abfgnprtw] [radius]`
`[radius]`	Radius to kill mobs in
`[-p]`	Also kill pets
`[-n]`	Also kill NPCs
`[-g]`	Also kill golems
`[-a]`	Also kill animals
`[-b]`	Also kill ambient mobs
`[-t]`	Also kill mobs with name tags
`[-f]`	Also kill all friendly mobs (Applies the flags *-abgnpt*)
`[-r]`	Also destroy armor stands
`[-w]`	Also kill water mobs

/brush paint

Description	Paint brush, apply a function to a surface
Permissions	worldedit.brush.paint
Usage	/brush paint <shape> [radius] [density] <forest|item|set>
<shape>	The shape of the region
[radius]	The size of the brush
[density]	The density of the brush

/brush none (or /brush unbind)

Description	Unbind a bound brush from your current item
Usage	/brush none

/brush clipboard (or /brush copy)

Description	Choose the clipboard brush
Permissions	worldedit.brush.clipboard
Usage	/brush clipboard [-abeov] [-m <sourceMask>]
[-a]	Don't paste air from the clipboard
[-v]	Include structure void blocks
[-o]	Paste starting at the target location, instead of centering on it
[-e]	Paste entities if available
[-b]	Paste biomes if available
[-m <sourceMask>]	Skip blocks matching this mask in the clipboard

/brush snowsmooth

Description	Choose the snow terrain softener brush Example: '/brush snowsmooth 5 1 -l 3'
Permissions	worldedit.brush.snowsmooth
Usage	/brush snowsmooth [radius] [iterations] [-l <snowBlockCount>] [-m <mask>]
[radius]	The radius to sample for softening
[iterations]	The number of iterations to perform
[-l <snowBlockCount>]	The number of snow blocks under snow
[-m <mask>]	The mask of blocks to use for the heightmap

/brush gravity (or /brush grav)

Description	Gravity brush, simulates the effect of gravity
Permissions	`worldedit.brush.gravity`
Usage	`/brush gravity [radius] [-h <height>]`
`[radius]`	The radius to apply gravity in
`[-h <height>]`	Affect blocks between the given height, upwards and downwards, rather than the target location Y + radius

`/brush heightmap`

Description	Heightmap brush, raises or lowers terrain using an image heightmap
Permissions	`worldedit.brush.heightmap`
Usage	`/brush heightmap [-efr] <imageName> [radius] [intensity]`
`<imageName>`	The name of the image
`[radius]`	The size of the brush
`[intensity]`	The intensity of the brush
`[-e]`	Erase blocks instead of filling them
`[-f]`	Don't change blocks above the selected height
`[-r]`	Randomizes the brush's height slightly.

`/brush extinguish` (or `/brush ex`)

Description	Shortcut fire extinguisher brush
Permissions	`worldedit.brush.ex`
Usage	`/brush extinguish [radius]`
`[radius]`	The radius to extinguish

`/brush feature`

Description	Feature brush, paints Minecraft generation features
Permissions	`worldedit.brush.feature`
Usage	`/brush feature <shape> [radius] [density] <type>`
`<shape>`	The shape of the region
`[radius]`	The size of the brush
`[density]`	The density of the brush
`<type>`	The type of feature to use

`/brush sphere` (or `/brush s`)

Description	Choose the sphere brush
Permissions	`worldedit.brush.sphere`
Usage	`/brush sphere [-h] <pattern> [radius]`
`<pattern>`	The pattern of blocks to set
`[radius]`	The radius of the sphere
`[-h]`	Create hollow spheres instead

`/brush raise`

Description	Raise brush, raise all blocks by one
Permissions	`worldedit.brush.raise`
Usage	`/brush raise <shape> [radius]`
`<shape>`	The shape of the region
`[radius]`	The size of the brush

`/brush smooth`

Description	Choose the terrain softener brush Example: '/brush smooth 2 4 grass_block,dirt,stone'
Permissions	`worldedit.brush.smooth`
Usage	`/brush smooth [radius] [iterations] [mask]`
`[radius]`	The radius to sample for softening
`[iterations]`	The number of iterations to perform
`[mask]`	The mask of blocks to use for the heightmap

`/brush cylinder` (or `/brush cyl`, `/brush c`)

Description	Choose the cylinder brush
Permissions	`worldedit.brush.cylinder`
Usage	`/brush cylinder [-h] <pattern> [radius] [height]`
`<pattern>`	The pattern of blocks to set
`[radius]`	The radius of the cylinder
`[height]`	The height of the cylinder
`[-h]`	Create hollow cylinders instead

`/brush set`

Description	Set brush, sets all blocks in the area
Permissions	`worldedit.brush.set`
Usage	`/brush set <shape> [radius] <pattern>`
`<shape>`	The shape of the region
`[radius]`	The size of the brush
`<pattern>`	The pattern of blocks to set

/brush apply

Description	Apply brush, apply a function to every block
Permissions	worldedit.brush.apply
Usage	/brush apply <shape> [radius] <forest|item|set>
<shape>	The shape of the region
[radius]	The size of the brush

/brush deform

Description	Deform brush, applies an expression to an area
Permissions	worldedit.brush.deform
Usage	/brush deform [-or] <shape> [radius] [expression]
<shape>	The shape of the region
[radius]	The size of the brush
[expression]	Expression to apply
[-r]	Use the game's coordinate origin
[-o]	Use the placement position as the origin

/brush lower

Description	Lower brush, lower all blocks by one
Permissions	worldedit.brush.lower
Usage	/brush lower <shape> [radius]
<shape>	The shape of the region
[radius]	The size of the brush

/brush erode

Description	Erode preset for morph brush, erodes blocks in the area
Permissions	worldedit.brush.morph
Usage	/brush erode [brushSize]
[brushSize]	The size of the brush

/brush dilate

Description	Dilate preset for morph brush, dilates blocks in the area
Permissions	worldedit.brush.morph
Usage	/brush dilate [brushSize]
[brushSize]	The size of the brush

/brush snow

Description	Snow brush, sets snow in the area
Permissions	`worldedit.brush.snow`
Usage	`/brush snow [-s] <shape> [radius]`
`<shape>`	The shape of the region
`[radius]`	The size of the brush
`[-s]`	Whether to stack snow

/brush biome

Description	Biome brush, sets biomes in the area
Permissions	`worldedit.brush.biome`
Usage	`/brush biome [-c] <shape> [radius] <biomeType>`
`<shape>`	The shape of the region
`[radius]`	The size of the brush
`<biomeType>`	The biome type
`[-c]`	Whether to set the full column

/brush morph

Description	Morph brush, morphs blocks in the area
Permissions	`worldedit.brush.morph`
Usage	`/brush morph [brushSize] [minErodeFaces] [numErodeIterations] [minDilateFaces] [numDilateIterations]`
`[brushSize]`	The size of the brush
`[minErodeFaces]`	Minimum number of faces for erosion
`[numErodeIterations]`	Erode iterations
`[minDilateFaces]`	Minimum number of faces for dilation
`[numDilateIterations]`	Dilate iterations

1.5.10 Biome Commands

/biomelist (or /biomels)

Description	Gets all biomes available.
Permissions	`worldedit.biome.list`
Usage	`/biomelist [-p <page>]`
`[-p <page>]`	Page number.

/biomeinfo

Description	Get the biome of the targeted block. By default, uses all blocks in your selection.
Permissions	`worldedit.biome.info`
Usage	`/biomeinfo [-pt]`
`[-t]`	Use the block you are looking at.
`[-p]`	Use the block you are currently in.

`//setbiome`

Description	Sets the biome of your current block or region. By default, uses all the blocks in your selection
Permissions	`worldedit.biome.set`
Usage	`//setbiome [-p] <target>`
`<target>`	Biome type.
`[-p]`	Use your current position

`//replacebiome`

Description	Replaces the biome of your current block or region. By default, uses all the blocks in your selection
Permissions	`worldedit.biome.set`
Usage	`//replacebiome [-p] <from> <target>`
`<from>`	The mask representing where to replace biomes
`<target>`	Biome type to set
`[-p]`	Use your current position

1.5.11 Chunk Commands

`/chunkinfo`

Description	Get information about the chunk you're inside
Permissions	`worldedit.chunkinfo`
Usage	`/chunkinfo`

`/listchunks`

Description	List chunks that your selection includes
Permissions	`worldedit.listchunks`
Usage	`/listchunks [-p <page>]`
`[-p <page>]`	Page number.

/delchunks

Description	Delete chunks that your selection includes
Permissions	`worldedit.delchunks`
Usage	`/delchunks [-o <beforeTime>]`
`[-o <beforeTime>]`	Only delete chunks older than the specified time.

1.5.12 Snapshot Commands

/restore (or //restore)

Description	Restore the selection from a snapshot
Permissions	`worldedit.snapshots.restore`
Usage	`/restore [snapshot]`
`[snapshot]`	The snapshot to restore

/snapshot (or /snap)

Description	Snapshot commands for restoring backups
Permissions	`worldedit.snapshots.restore`, `worldedit.snapshots.list`
Usage	`/snapshot <before|use|sel|after|list>`

/snapshot before

Description	Choose the nearest snapshot before a date
Permissions	`worldedit.snapshots.restore`
Usage	`/snapshot before <date>`
`<date>`	The soonest date that may be used

/snapshot use

Description	Choose a snapshot to use
Permissions	`worldedit.snapshots.restore`
Usage	`/snapshot use <name>`
`<name>`	Snapshot to use

/snapshot sel

Description	Choose the snapshot based on the list id
Permissions	`worldedit.snapshots.restore`
Usage	`/snapshot sel <index>`
`<index>`	The list ID to select

`/snapshot after`

Description	Choose the nearest snapshot after a date
Permissions	`worldedit.snapshots.restore`
Usage	`/snapshot after <date>`
`<date>`	The soonest date that may be used

`/snapshot list`

Description	List snapshots
Permissions	`worldedit.snapshots.list`
Usage	`/snapshot list [-p <page>]`
`[-p <page>]`	Page of results to return

1.5.13 Scripting Commands

`/cs`

Description	Execute a CraftScript
Permissions	`worldedit.scripting.execute`
Usage	`/cs <filename> [args...]`
`<filename>`	Filename of the CraftScript to load
`[args...]`	Arguments to the CraftScript

`/.s`

Description	Execute last CraftScript
Permissions	`worldedit.scripting.execute`
Usage	`/.s [args...]`
`[args...]`	Arguments to the CraftScript

1.5.14 Utility Commands

`//fill`

Description	Fill a hole
Permissions	`worldedit.fill`
Usage	`//fill <pattern> <radius> [depth]`
`<pattern>`	The blocks to fill with
`<radius>`	The radius to fill in
`[depth]`	The depth to fill

`//fillr`

Description	Fill a hole recursively
Permissions	`worldedit.fill.recursive`
Usage	`//fillr <pattern> <radius> [depth]`
`<pattern>`	The blocks to fill with
`<radius>`	The radius to fill in
`[depth]`	The depth to fill

`//drain`

Description	Drain a pool
Permissions	`worldedit.drain`
Usage	`//drain [-w] <radius>`
`<radius>`	The radius to drain
`[-w]`	Also un-waterlog blocks

`/fixlava` (or `//fixlava`)

Description	Fix lava to be stationary
Permissions	`worldedit.fixlava`
Usage	`/fixlava <radius>`
`<radius>`	The radius to fix in

`/fixwater` (or `//fixwater`)

Description	Fix water to be stationary
Permissions	`worldedit.fixwater`
Usage	`/fixwater <radius>`
`<radius>`	The radius to fix in

`/removeabove` (or `//removeabove`)

Description	Remove blocks above your head.
Permissions	`worldedit.removeabove`
Usage	`/removeabove [size] [height]`
`[size]`	The apothem of the square to remove from
`[height]`	The maximum height above you to remove from

`/removebelow` (or `//removebelow`)

Description	Remove blocks below you.
Permissions	`worldedit.removebelow`
Usage	`/removebelow [size] [height]`
`[size]`	The apothem of the square to remove from
`[height]`	The maximum height below you to remove from

`/removenear` (or `//removenear`)

Description	Remove blocks near you.
Permissions	`worldedit.removenear`
Usage	`/removenear <mask> [radius]`
`<mask>`	The mask of blocks to remove
`[radius]`	The radius of the square to remove from

`/replacenear` (or `//replacenear`)

Description	Replace nearby blocks
Permissions	`worldedit.replacenear`
Usage	`/replacenear <radius> [from] <to>`
`<radius>`	The radius of the square to remove in
`[from]`	The mask matching blocks to remove
`<to>`	The pattern of blocks to replace with

`/snow` (or `//snow`)

Description	Simulates snow
Permissions	`worldedit.snow`
Usage	`/snow [-s] [size] [height]`
`[size]`	The radius of the cylinder to snow in
`[height]`	The height of the cylinder to snow in
`[-s]`	Stack snow layers

`/thaw` (or `//thaw`)

Description	Thaws the area
Permissions	`worldedit.thaw`
Usage	`/thaw [size] [height]`
`[size]`	The radius of the cylinder to thaw in
`[height]`	The height of the cylinder to thaw in

`/green` (or `//green`)

Description	Converts dirt to grass blocks in the area
Permissions	`worldedit.green`
Usage	`/green [-f] [size] [height]`
`[size]`	The radius of the cylinder to convert in
`[height]`	The height of the cylinder to convert in
`[-f]`	Also convert coarse dirt

`/extinguish` (or `//ex`, `//ext`, `//extinguish`, `/ex`, `/ext`)

Description	Extinguish nearby fire
Permissions	`worldedit.extinguish`
Usage	`/extinguish [radius]`
`[radius]`	The radius of the square to remove in

`/butcher`

Description	Kill all or nearby mobs
Permissions	`worldedit.butcher`
Usage	`/butcher [-abfgnprtw] [radius]`
`[radius]`	Radius to kill mobs in
`[-p]`	Also kill pets
`[-n]`	Also kill NPCs
`[-g]`	Also kill golems
`[-a]`	Also kill animals
`[-b]`	Also kill ambient mobs
`[-t]`	Also kill mobs with name tags
`[-f]`	Also kill all friendly mobs (Applies the flags *-abgnpt*)
`[-r]`	Also destroy armor stands
`[-w]`	Also kill water mobs

`/remove` (or `/rem`, `/rement`)

Description	Remove all entities of a type
Permissions	`worldedit.remove`
Usage	`/remove <remover> <radius>`
`<remover>`	The type of entity to remove
`<radius>`	The radius of the cuboid to remove from

`//calculate` (or `//calc`, `//eval`, `//evaluate`, `//solve`)

Description	Evaluate a mathematical expression
Permissions	`worldedit.calc`
Usage	`//calculate <input...>`
`<input...>`	Expression to evaluate

`//help`

Description	Displays help for WorldEdit commands
Permissions	`worldedit.help`
Usage	`//help [-s] [-p <page>] [command...]`
`[-s]`	List sub-commands of the given command, if applicable
`[-p <page>]`	The page to retrieve
`[command...]`	The command to retrieve help for

1.6 Usage

WorldEdit has many commands, tools, and features. For a listing of commands, see the *commands* page (or just use `//help` in-game). These pages will go more in depth on how to use the various features

1.6.1 General

History

Before you dive in to using WorldEdit, you should know how to use history to undo mistakes you may make along the way. Your history is stored in your session, more details *here*.

Undo and Redo

Every action taken with WorldEdit, whether performed by a command or a tool, will be stored in your session's history. By default, WorldEdit stores your last 15 actions (this can be changed in the *configuration*).

If you ever want to undo an action, you can use the `//undo` command. This will undo your last action. To redo it, use `//redo`.

Warning

WorldEdit only records direct changes. Indirect changes such as liquids that start flowing, attached blocks (doors, signs, tall grass) popping off, and so on, will not be recorded by history. Be careful while working with these things.

As shown in the *quick start*, history commands can operate on multiple actions at once. You can use `//undo 5`, for example, to undo your last 5 actions.

History commands also let players will permission undo or redo the actions of other players when on multiplayer. Note that thanks to sessions, you can undo another player's actions for up to 10 minutes after they've logged out (and for as long as they are logged in). To do this, use `//undo <number> <player name>`.

Your stored history can be cleared with the `/clearhistory` command.

Sessions

When you select a region or change your preferences in-game, your information will be put into a session that will be kept active as long as you stay logged in. Upon disconnecting, your session will be discarded in 10 minutes, allowing you to log back in and retain your session. Each person's session is separate when connected to a server.

Sessions contain various things such as the following:

- Your current selection
- Your history
- Your block change limit
- Your selected snapshot to restore from
- Your clipboard
- Any currently bound tools/brushes

Warning

WorldEdit drops *all* sessions in single-player when leaving a world, so it is not possible to directly copy between worlds or retain most tool bindings after logging out in single-player.

Persistent Data

Some of the data in a session is *persistent*, or stored offline in the server files. This includes the following:

- Your selection wand binding
- Your navigation wand binding
- If server-side CUI is currently on
- The last script you used
- Your default selector

Patterns

You may have noticed (or if you haven't yet, you soon will) that many WorldEdit *commands* take a "pattern" as a parameter. Patterns range from very simple (such as a single block - `stone`) to very complex. Patterns determine what blocks get set into the world as a command, tool, etc operates.

Available Patterns

Note

This list may be incomplete as patterns are added to WorldEdit. In addition, our API allows other plugins to register new patterns, which will not be listed here.

Tip

Here's a video detailing some of these patterns which were added in WorldEdit 7: https://www.youtube.com/watch?v=S5wCVMf3SvM

Single Block Pattern

The most basic pattern of just a single block. A block is identified by three parts: the block type, additional block states, and nbt data. These three links to the Minecraft Wiki, along with WorldEdit's in-built tab-completion for commands, should guide you in specifying the block you want. Additional states are always appended to the type using the syntax `block_type[state1=value,state2=value,...]`. Note that when states are not specified, or if some are left out, the default values will be used for those states. NBT is written using Minecraft's SNBT syntax, `block_type{'nbt_key':'value'}`. NBT data always comes after the block states if supplied, `block_type[property=value]{'nbt_key':'value'}`.

Example: Single block patterns

Setting a selection to stone:

```
//set stone
```

Setting a selection to a note block with a specific instrument and pitch:

```
//set note_block[instrument=chime,note=18]
```

Setting a selection to an oak stair block facing east with the bigger part on top:

```
//set oak_stairs[facing=east,half=top]
```

Setting a selection to an oak sign that is waxed:

```
//set oak_sign{'is_waxed':1}
```

Setting a selection to an oak sign that is waxed and rotated:

```
//set oak_sign[rotation=12]{'is_waxed':1}
```

Random Pattern

This pattern allows setting random blocks from any number of other patterns. The basic form is as simple as a comma-separated list of patterns, which will be chosen from evenly. You can also specify weights for each pattern with `<x>%<pattern>`.

Example:: Random Patterns

Setting a selection to different types of stone, equally distributed:

```
//set stone,diorite,andesite,granite
```

Setting a selection to mostly red wool, with a small amount of glass (5:1 ratio, see note below):

```
//set 50%red_wool,10%red_stained_glass
```

Note

Despite the percentage sign, weights need not add up to 100. They are cumulative and will be divided by the total. That is, `5%dirt,15%stone` means 25% of blocks will be dirt, and 75% will be stone. In other words, weights are relative to each other, not to 100. Because of that, the pattern `5%dirt` isn't valid. If you only want to set 5% of blocks to dirt, you should use the *random noise mask*.

Tip

You can use any other pattern as one of the choices, not just the single block pattern. Keep reading to see more patterns…

Random State Pattern

Prefixing any block type with an asterisk (*) will randomly choose between all states for that block for each position.

Example: Random State Pattern

Setting oak logs facing in random directions:

```
//set *oak_log
```

Clipboard Pattern

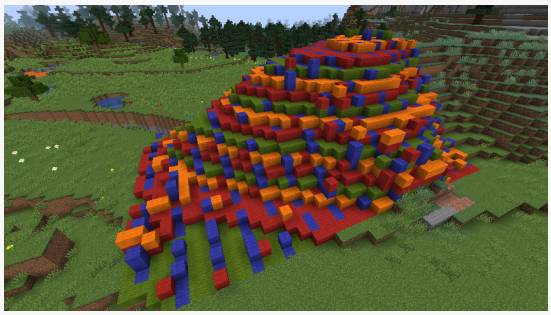
The `#clipboard` pattern will take blocks from your *clipboard* in the same arrangement. This makes it easy to build one part of a repeating complicated pattern by hand, and then repeat it over and over. You can also offset the pattern by adding `@[x,y,z]`.

Example: Using the clipboard pattern

Replacing *all existing blocks* to your clipboard:

```
//replace #existing #clipboard
```

Using the clipboard in the first image to replace a hill. Note the repeating layers.



Using an offset to align the clipboard:

```
//set #clipboard@[2,0,1]
```

Type or State Applying Pattern

This pattern, prefixed by `^`, lets you set the type or states of a block without modifying everything else. This pattern will, for example, allow you to change a spiral staircase from oak to acacia without having to worry about the stairs facing in different directions and so on. You can either specify a block type (to change block type but not states, where applicable), or any number of states (to only change those states, where applicable).

Example: Type/State Applying Patterns

Replacing all oak stairs to acacia stairs, while maintaining orientation, etc:

```
//replace oak_stairs ^acacia_stairs
```

Removing the water from all waterloggable blocks:

```
//set ^[waterlogged=false]
```

Doubling up all slabs:

```
//replace ##slabs ^[type=double]
```

Replacing all stairs with a random selection of stairs, while maintaining orientation, etc:

```
//replace ##stairs ^##stairs
```

Block Category Pattern

This pattern allows setting random blocks within a block category, often referred to as a "tag". Tags allow grouping blocks together under a single name. Minecraft comes with many tags inbuilt (see the link) and also allows creating and modifying tags via data packs. You may already have noticed these tags being used as a *mask* in the example above (`##slabs`).

The syntax for this pattern is `##<tag name>`, which will randomly choose between the default state of all blocks in the category. You can also mix this with the random state pattern (`##*<tag name>`) to use all states, not just the defaults.

Example: Block Category Pattern Usage

Replacing all existing blocks with rainbow wool:

```
//replace #existing ##wool
```

Setting the selection to random types of slabs, both top/bottom/double, and waterlogged at random:

```
//set ##*slabs
```

Special Block Data Syntax

Some blocks have additional syntax for setting extra information.

Sign Text

You can set text on signs by separating it with a pipe symbol (`|`). Note that if the text has spaces, you must wrap the entire pattern in quotes `""`.

Example: Setting sign text

Simple Example:

```
//set oak_sign|Line1|Line2
```

With spaces and rotation:

```
//set "oak_wall_sign[facing=north]|Hello world|Second|Third line"
```

Player Heads

You can set the skin of a player head by specifying a username after the pipe symbol.

Example: Setting a skin on a head

```
//set player_head|dinnerbone
```

Mob Spawners

You can set the type of mob to be spawned (again via the pipe symbol). Note that the name of the mob must be an entity ID. Prefixing *minecraft:* is optional, modded mobs must have a namespace.

Example: Creating a squid spawner

```
//set spawner|squid
```

Masks

Masks, alongside *patterns*, are commonly used in WorldEdit commands. Unlike patterns, masks determine *which* blocks will be affected by commands, brushes, and so on.

Aside from commands that take a mask as a parameter (such as `//replace [mask] <pattern>`), you can also apply masks to individual *brushes* by using the `/mask` command while holding the brush, or you can apply a mask to all your WorldEdit actions globally with `//gmask`.

Note

Masks applied through different means will stack with each other. If you set your global mask with `//gmask dirt`, and then set a brush mask with `/mask stone`, that brush will not be able to modify any blocks at all! This is because the combined mask will only match blocks which are both stone *and* dirt!

Tip

You can clear your global mask by using `//gmask` again without arguments.

Combining Masks

To get a mask which matches the *intersection* of multiple masks, use a space to separate them. The intersection will match when *all* of the given masks match.

Example: Combining Masks

Replacing surface stone with dirt using a mask intersection:

```
//replace "stone <air" dirt
```

Setting a mask which restricts the current brush to modifying air blocks in your selection:

```
/mask "air #sel"
```

Available Masks

Block Mask

The simplest of masks, the block mask matches one or more blocks or block states. Just like the *single block pattern*, you can specify either a block type alone, or a block type with any number of states specified. Unlike the pattern, masks will *not* use default values for unspecified block states - they will "fuzzy" match any value of the unspecified state(s).

To match more than one block, separate each with a comma.

Example: Using the block mask

Removing all oak fences from your selection:

```
//replace oak_fence air
```

Removing all oak fences that are attached on the east side, and oak fence gates:

```
//replace oak_fence[east=true],oak_fence_gate air
```

Mask Negation

The `!` symbol can be used to negate everything that comes after it. That is, it matches anything *not* matched by a different mask. Any other mask can follow this.

Example: Negating a mask

Replace any block that isn't dirt, stone, or grass with stone:

```
//replace !dirt,stone,grass_block stone
```

Existing Block Mask

The mask `#existing` will match any blocks that aren't air. Note that this isn't the same as *!air*, since the game actually has multiple types or air that it uses in some cases.

Solid Block Mask

The mask `#solid` will match any blocks that are considered "solid". That is - blocks that restrict entities (such as players) from moving through them.

Full Cube Mask

The mask `#fullcube` will match any blocks that take up the entire block cube.

Offset Mask

Using > (overlay) or < (underlay) preceding another mask will match blocks that are above or below the other mask, respectively.

Example: Offset masks

Creating a layer of slabs above planks in your selection:

```
//replace >##planks smooth_stone_slab
```

Adjacency Mask

Using ~ preceding another mask will match blocks that are adjacent to the other mask.

Note

Adjacency for the case of this mask is defined as being directly next to or above/below. Diagonal blocks do not match.

Example: Adjacency masks

Surrounding all mob spawners with glowstone:

```
//replace ~spawner glowstone
```

Region Masks

While it doesn't make sense for commands like replace, setting a region mask can be very useful for using brushes inside a limited area. For example, if you want to brush some dirt around the base of your wall, you can select the wall, and then negate a region mask so that the dirt doesn't affect the wall (but still affects the ground around it).

The first type of region mask is `#region` (aliases: `#sel`, `#selection`), which will make a copy of your region at the time you run the command and use that as the mask.

The second type of region mask is `#dregion` (d for dynamic, also `#dsel`, `#dselection`) which will always stay updated with your current selection.

Clipboard Mask

The mask `#clipboard` will match any blocks that match the current clipboard. This can be paired with commands such as *//regen -c* ` to match all blocks that are the same as natural generation.

Block Category Mask

Block categories, or tags can also be used as masks. A category mask will match any block that is in that category. Just like the pattern, the syntax is *##<tag>*.

Example: Block Category Masks

Replacing all carpets with a layer of snow:

```
//replace ##carpets snow
```

Random Noise Mask

The noise mask can create random noise. Specifying `%<percent>` will match the given percentage of blocks. Unlike the weighted patterns, `%50` is actually 50% of blocks.

Note

Make note of the syntax difference here; Unlike patterns, the % sign preceeds the desired percentage for this mask

Example: Using the random noise mask

Randomly replacing 50% of your selection with stone:

```
//replace %50 stone
```

Block State Mask

Like the block mask, this mask matches block states. Unlike the block mask, you don't need to specify a block type. This means you can match any block that has a property in a given value.

The state mask has two modes, lenient and strict. In lenient mode (`^[state=value,...]`, it will match any block that has the given block states equal the given value, *or* any block that does not even have those properties. In strict mode (`^=[state=value,...]`), it will *only* match blocks that have the block states equal to that value.

Example: Using the block state mask

Removing all closed door, gates, and trapdoors:

```
//replace ^=[open=false] air
```

Expression Mask

This mask can evaluate a mathematical expression upon each block. The mask starts with = and then must have an *expression* which can use the variables `x`, `y`, and `z`. The mask will match if the expression returns a positive value.

Example: Expression masks

Only edit blocks below a certain y-level:

```
//gmask =y<64
```

Only edit blocks with air two blocks below:

```
//gmask =queryRel(0,-2,0,0,0)
```

Biome Mask

The biome mask matches blocks with the given biome. Its syntax is `$<biome id>`. The biome ID must be the namespaced id, with *minecraft:* being optional for vanilla biomes, and mod ids being required for mod-added biomes.

Surface Mask

The surface mask matches blocks that are exposed to air on at least one face. This means it'll match only the surface of an object, nothing fully occluded by other blocks.

The mask is `#surface` (alias `#exposed`).

1.6.2 Navigation

You may often find yourself in need of getting to places in order to do your work better. The following commands deal with that issue.

The Navigation Wand

WorldEdit gives you access to a navigation wand, a *tool* that is by default bound to the compass (though you can *configure* this).

At any time, you can left click while holding a compass to jump to the block you are looking at (as with the jumpto command, described below). You can also right click with the compass to pass through walls (as with the thru command, described below).

Freeing yourself

```
/unstuck
/!
```

This will get you out of a block if you somehow got embedded inside. It moves you to the top-most free position, choosing to do nothing if you were not stuck to begin with. The short alias (`/!`) can often be useful if you aren't in creative mode and find yourself suffocating after pasting or generating something on top of yourself.

Jumping to the block in sight

```
/jumpto
```

This command sends you to the top of the block that you are looking at. If that block is a vertical wall, you will be placed on top of the cliff at the very edge.

Passing through walls

```
/thru
```

This command sends you through a wall in the direction you are looking. Just look into a wall and use the command. Make sure that you do not look downwards into a wall because it will attempt to go through the ground. This command limits the thickness of the wall to a reasonable amount.

Ascending and descending

```
/ascend [levels]
/descend [levels]
```

These two commands allow you to pass through the ceiling above or floor below you. For example, if you were inside a home, you would move to the roof of the building if you used /ascend. You can also optionally provide a number of levels to ascend or descend. For example, if you were in a skyscraper on the bottom floor, and used `/ascend 2`, you'd be on the 3rd floor.

Ascending to the ceiling

```
/ceil [clearance]
```

This command brings you up to the ceiling of the current room that you are in. If you provide no clearance parameter to use, you will be right up to the ceiling. If you do specify a clearance, the space above your head will be larger. A glass block is placed under your feet in order to support you. You must remove the block manually.

Ascending up an arbitrary distance

```
/up <distance>
```

This will move you up a certain number of blocks. You cannot pass through walls with this command and a glass block will be placed at your feet to support you. The glass block has to be removed manually after you are done.

1.6.3 Regions

Selection

A fundamental part of WorldEdit is working with a selection or region. For example, if you wanted to replace all the glass blocks inside a square area with dirt, you would have to tell WorldEdit where this square is. WorldEdit gives you several ways to select a region for editing, this section will introduce you to selecting regions and show you what you can do with them.

- *Want to see selection lines?*
- *Selecting Cuboids*

Want to see selection lines?

To see lines showing your selection, you can either:

1. Access a limited version of the selection outlines server-side via the `//drawsel` command. It works only for cuboid selections that are not larger than 48x48x48 (or 32x32x32 on older versions), and you have to be in creative mode. These limitations are due to how structure blocks have worked in Minecraft for a long time.
2. Use a third party client-side mod, e.g. WorldEdit CUI (Fabric). Note that this mod requires Fabric, so you will have to install it at first.

Note

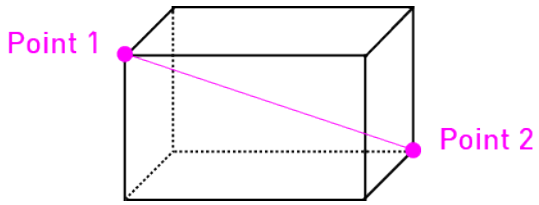
If you would like to use an older version of Minecraft (1.12 or earlier), in addition to downloading an older WorldEdit (version 6), you may also need the old WorldEditCUI mod by Mumfrey. Note that this mod requires LiteLoader (installation instructions on that page) instead.

Selecting Cuboids

WorldEdit allows you to select **cuboids** (think a 3D rectangle) by choosing two points at two corners of the cuboid. The diagram below indicates how two points can form a cuboid. The cuboids you select can only be aligned to the world (they cannot be rotated at an angle).

Tip

You can also select other shapes such as 2D polygons, ellipsoids, spheres, cylinders, and more. Information on these other selection modes is described later in this page.



There are several different ways of choosing these two points and you can mix and match.

Selecting with the wand

```
//wand
```

The most intuitive way to select a region is by using wand. To get the wand, use `//wand` (it is, by default, a wooden axe). **Left clicking** a block with the wand marks that block as the first corner of the cuboid you wish to select. A **right-click** chooses the second corner.

You can bind the selection wand to a different item either by changing the *configuration* or using the `/tool selwand` command. In this regard, it is a *tool*.

Selecting at your own location

```
//pos1
//pos2
```

These commands set the first and second corners to the block above the one that you are standing on. Generally the wand suffices for most tasks and you likely will not need to use this. This is useful if you are flying in mid-air and don't have blocks you can click with the wand.

Selecting a specific location

```
//pos1 x,y,z
//pos2 x,y,z
```

These commands set the first and second corners to the block specified by the given coordinates. This is useful for when you have pre-calculated specific coordinates that you want to manipulate.

Selecting with your crosshair

```
//hpos1
//hpos2
```

These commands set the first and second corners to the block that you are looking at. This allows you to select points from far away and make particularly large cuboid regions with ease.

Selecting the current chunk

```
//chunk
```

This command selects all the blocks in the chunk that you are standing in. Chunks are 16 by 16 and are 256 blocks high. Using `//chunk -s` instead will select *all* chunks that your current selection intersects.

Adjusting the selection

Expanding the selection

```
//expand <amount> [direction]
//expand <amount> <reverse-amount> [direction]
//expand vert
```

This command allows you to easily enlarge a region in several different ways:

- By specifying a direction
- By looking in the direction (only for cardinal directions)
- To the sky and to bedrock (using `vert`)

To specify a direction, use "N", "S", "W", "E", "U" (for up), or "D" (for down) for the direction. If you want to merely look in the direction, either use "me" for the direction or don't enter a direction parameter.

You can also specify relative directions like “F” (forward, same as “me”), “B” (back), “L” (left), “R” (right) which will go be relative to the direction you are facing.

You can also specify multiple directions separated by commas to expand in multiple directions at once, such as `//expand 10 n,w`.

You can specify two numbers and the region will be expanded in two opposite directions simultaneously.

Example: Expanding upwards

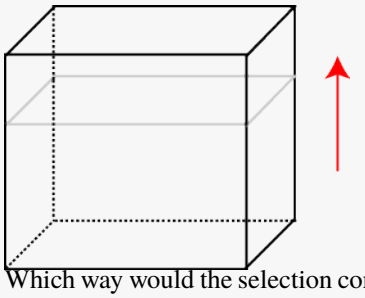
For example, if you used `//expand 10 up`, the selection would grow larger at the top as shown.

Contracting the selection

```
//contract <amount> [direction]
//contract <amount> <reverse-amount> [direction]
```

This command works similarly to `//expand`, but instead contract.

Example: Contracting downwards



Which way would the selection contract? If you used `//contract 10 down`, then the selection would shrink from the top.

Inset and Outset

Tip

If you want to expand or contract in all directions at once (or just horizontal or vertical directions), you can use the `//outset` and `//inset` commands. For example, `//outset -v 5` will expand your selection vertically (both up and down) 5 blocks each, while `//inset -h 5` will contract your selection horizontally (north, west, south, and east) 5 blocks each. Leaving out the `v` or `h` will work in all 6 directions.

Trimming the selection

```
//trim [mask]
```

This command minimises the selection to the smallest size necessary to still fit all blocks that match the given mask. When no mask is specified, it will trim the selection to fit non-air blocks as if it were using the `#existing` mask.

Selection Information

WorldEdit also provides you with commands for getting information about the area you have selected.

Getting selection size

```
//size [-c]
```

Gets the area and dimensions of your selection.

The `-c` flag will instead measure your clipboard size instead of your selection.

Counting block frequency

```
//count <mask>
```

Counts the number of blocks in your selection which match the mask.

Finding the block distribution

```
//distr [-cd]
```

Shows the block distribution in the selection area.

The `-c` flag operates on your clipboard instead of your selection.

The `-d` flag separates by block states instead of just types.

Example: Block distribution output

Only by block type:

```
75      (52.083%) Air (minecraft:air)
41      (28.472%) Grass Block (minecraft:grass_block)
18      (12.500%) Rose Bush (minecraft:rose_bush)
5       (3.472%) Grass (minecraft:grass)
```

Separating by states:

```
75      (52.083%) Air (minecraft:air)
41      (28.472%) Grass Block (minecraft:grass_block[snowy=false])
9       (6.250%) Rose Bush (minecraft:rose_bush[half=upper])
9       (6.250%) Rose Bush (minecraft:rose_bush[half=lower])
5       (3.472%) Grass (minecraft:grass)
```

Selection Modes

Using the `//sel <mode>` command allows you to change between different shapes. It is recommended to install the CUI mod when using more complex shapes so you can visualize what you're selecting.

`//sel cuboid`

The standard cuboid selection mode, described above.

`//sel extend`

Left-click to select first point. All subsequent points are selected by right-clicking. Every right-click will extend the cuboid selection to encompass the new point.

`//sel poly`

Left-click to select first point. All subsequent points are selected by right-clicking. Every right-click will add an additional point. The top and bottom will always encompass your highest and lowest selected points.

`//sel ellipsoid`

Left-click to choose center, right-click to extend. You can control the radii along the x, y, and z planes individually depending on where you click.

`//sel sphere`

Left-click to select center, right-click to extend. Selection will always be a sphere from the first point which has a radius to the second point.

`//sel cyl`

Left-click to choose center, right click to extend. You can control the x and z radii, while the height will always encompass your highest and lowest points.

`//sel convex`

Left-click to select first point. All subsequent points are selected by right clicking. The selection is a convex hull encompassing all your selected points.

Region Operations

Once you have *selected a region*, it's time for the real fun to begin.

- *Regenerating*
- *Naturalizing*
- *Placing flora*
- *Generating forests*
- *Hollowing areas*
- *Creating lines and curves*
- *Setting a block in the center*
- *Deforming regions*
- *Setting Biomes*

Setting Blocks

The most basic operation, the `//set <pattern>` command allows you to set all blocks in your selection to a given pattern.

Example: Setting your selection

A simple pattern:

```
//set stone
```

Complex *patterns*:

```
//set 30%red_wool,*oak_log,#copy,##slabs
```

Replacing Blocks

If you don't want to set *all* the blocks in your selection, you can decide which ones should be affected by specifying a mask with the `//replace [mask] <pattern>` command.

Note that you can omit the mask argument in the command - it will default to the *existing block mask*.

Example: Replacing blocks in your selection

Replacing all non-air blocks with grass:

```
//replace grass_block
```

Replacing all stone with green_wool:

```
//replace stone green_wool
```

Replacing surface sand with metal blocks:

```
//replace "sand <air" diamond_block,iron_block,gold_block
```

Building walls (and other outlines)

Sometimes, you want to create walls or other hollow shapes. There are two main ways to do this:

For cuboid selections, the `//faces <pattern>` command (alias: `//outline`) will allow you to fill all 6 faces (up, down, north, south, east, west). Using this command on any other type of selection will create a bounding box (i.e. a cuboid region that entirely encompasses whatever shape you had selected) and create the faces of that cuboid.

For any type of selection, the `//walls <pattern>` command will allow you to make a hollow shell of the selection without a ceiling and floor. The specifics of this depends on the selection type.

Overlaying

The `//overlay <pattern>` command allows you to overlay blocks in the selection with another block (or pattern). Note that the top of your selection must be "open" - this command will look downwards from the block above your selection in every column until it hits a non-air block, and then place your pattern above that. This is useful for overlaying torches, fences/walls, road blocks, etc. on top of existing terrain.

Stacking

The `//stack <count> [direction]` command allows you to repeat your selection a number of times in a given direction. This will essentially copy and paste your selection over and over.

If a direction is not specified, it will stack in the direction you are facing.

Some potential uses for this include:

- Extending bridges
- Making tunnels
- Repeating a segment of a hand-built structure

There are several flags available:

- `-m <mask>` will set a source *mask*, only stacking matching blocks
- `-e` will also copy entities into each stacked area
- `-b` will also copy biomes into each stacked area (you may need to re-join the world to see changes)

Some uses of the stack command

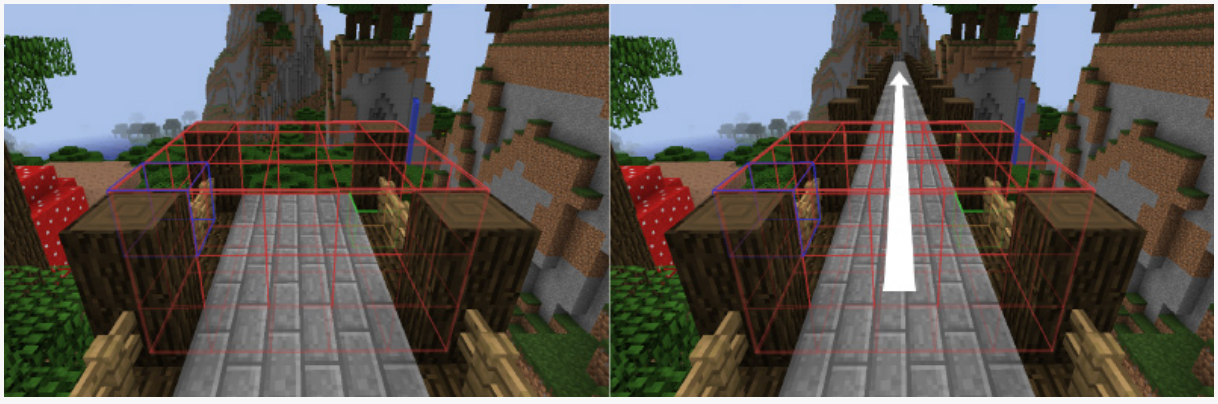


Fig. 1: Bridge extended with `//stack`

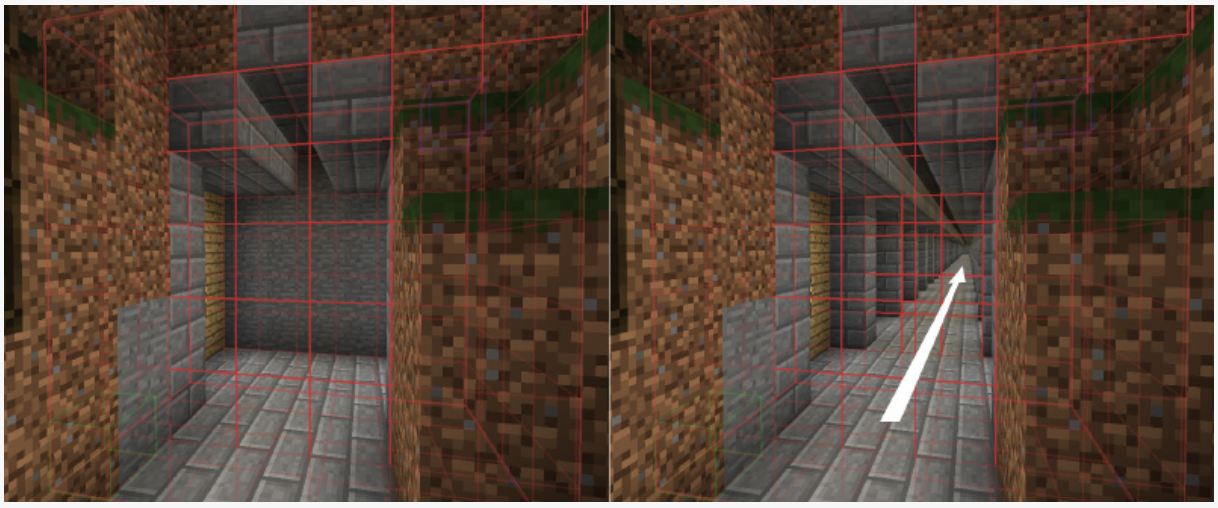


Fig. 2: Digging a tunnel through a mountain effortlessly. Be sure to select the ceiling and floor blocks.

Moving

```
//move <distance> [direction] [fill pattern]
```

If you've built something, only to find out that you need to move it a little to the side, this command can be very helpful as it will shift the entire area like magic. The command takes a distance to move the area, an optional direction, and also a block to fill with the existing area now left void by the move.

The direction parameter works like that of the `//stack` command: use any direction, defaulting to *"me"* - your current heading.

If you don't provide a block to fill with, the area left behind will be filled with air.

You can also use the `-s` flag to move your selection along with the blocks, so your new selection will be in the same place as the moved blocks.

The move command also takes the same three flags as `//stack`, `-e` to move entities, `-b` to copy biomes (source biomes unaffected), and `-m <mask>` to move only matching blocks.

Example: Using the move command

Moving the selection 2 blocks forward, leaving stone:

```
//move 2 me stone
```

Moving the selection 5 blocks down, leaving air behind:

```
//move 5 down
```

Smoothing

The `//smooth [iterations]` command will smooth terrain out. You can increase the number of iterations to make areas more smooth. Note that this command works with a heightmap, and while it excels at smoothing out surface terrain, it it not suitable for smoothing caves, walls, or objects.

Regenerating

The `//regen` command will regenerate your selection to its state when the world was freshly generated. It is based on the world's current world generator and seed, meaning running it multiple times will produce the same results.

> **Warning**
>
> The regen command will use the current world generator, which means if the world was generated via an external tool, or if Minecraft's terrain generation has changed in the meantime, the regenerated area will not match everything around it. If possible, consider taking a backup of your entire world ahead of time for use with *snapshots*.

Naturalizing

The `//naturalize` command will naturalize terrain by creating a layer of grass, followed by layers of dirt and then stone.

Fig. 3: Making the land look natural.

Placing flora

The `//flora` command will scatter tall grass and flowers over grass, and cacti and dead grass on sand in your selection. It works similarly to overlay, if you need a more complex pattern.

Generating forests

The `//forest <tree type> [density]` command will plant a forest with trees of your choosing. The density must be a number between 0 and 100, and controls how often WorldEdit will try to plant a tree within the area. The default is 5.

Hollowing areas

Using `//hollow [thickness] [fill pattern]` command will hollow out objects in your selection, leaving a shell with the given thickness. By default, the interior of the hollowed object will be filled with air, unless you specify something else.

Creating lines and curves

When you have a cuboid region selected, you can draw a line between the first and second points you selected. The command `//line <pattern> [thickness]` will create a line of the given pattern and thickness, and adding the `-h` flag will make it hollow, allowing you to generate a "tube".

To make a curve with more points, use the `//sel convex` selection mode to select multiple points. Then use `//curve <pattern> [thickness]` to draw a spline through all the points selected (in order!) of the given pattern and thickness - again, `-h` will make it hollow.

Setting a block in the center

The `//center <pattern>` command will set the center block (or 2 blocks, along any axis of even length) of your selection.

Deforming regions

Using `//deform <expression>`, you can apply a *custom expression* to all blocks in your selection.

The expression should take the variables `x`, `y`, and `z` and change them to the *new* coordinate that should be copied to the current x/y/z. For example, `y-=1` will move every block up one, since each block will be copied from the block *below* it.

By default, coordinates are normalized from -1 to 1 on each axis, from the min to max points of your selection. Using the `-r` flag will use raw world coordinates, while `-o` will use Minecraft coordinates scale offset to your placement position.

Example: Deforming regions

Making bumpy terrain:

```
//deform y+=0.2*sin(x*10)
```

Flipping your selection on its side:

```
//deform swap(x,y)
```

Setting Biomes

While WorldEdit mostly focuses on manipulating blocks, the `//setbiome <biome>` command allows you to set the biome in your selection. The biome type should be specifed by biome id. If you are using a biome added by a mod, the `namespace:` must prefix the id, e.g. `minecraft:plains`.

Note

WorldEdit supports Minecraft's 3D biomes since 7.2, on 1.15 and above. This means that for the most part, the biome will be set inside your selection only, even in the Y direction. Unfortunately, Mojang made biomes 4x4x4 cubes instead of 1x1x1 like blocks, and it also fuzzes on the edges. This means that biomes can't be as finely controlled as before, and that it can't be constrained to your selection entirely.

Additionally, until 1.18, Mojang made the overworld not entirely respect 3D biomes, so in some cases WorldEdit will also set the biome at `Y = 0` to ensure that spawning and visuals work properly.

To more closely emulate the old behavior of setting columns, use `//expand vert` before setting biomes.

1.6.4 Clipboard

WorldEdit has a powerful clipboard function that allows you to copy an area, paste it, and even save it to and load it from files. Clipboard contents are currently only cuboids and copying use the region you have selected.

Note that like *history*, your current clipboard is stored in your session and thus will be kept for 10 minutes after logging off (of a server).

Also like history, your current clipboard can be cleared with the */clearclipboard* command.

Copy and cut

The `//copy` command copies your current selection to your session's clipboard, **keeping track of where you are relative to the copy**. The second part of that sentence is very important; if you want to later paste, for example, a bridge so that it is under where you are standing, you must stand in a location above the bridge when you make the copy. This method allows you to easily align your later paste because you can plan ahead a bit; it requires some spatial abilities to master the copying process but you will find it particularly helpful once you get the hang of it.

`//cut` works just like `//copy` except that it also deletes the selected area afterwards. By default, it leaves air, but you can also specify a different block to leave behind.

> **Note**
>
> This remembers your **current position relative to the copy**. This is a very important concept to grasp otherwise you will not be able to control where you paste your copy!

Both commands have three additional flags:

- `-e` can be specified to also copy/cut entities from the selection
- `-b` can be specified to also copy biomes from the selection ("cutting" selections doesn't make sense - some biome needs to be left there)
- `-m <mask>` can be used to specify a *mask* of blocks to copy/cut. Any blocks that do not match will be replaced with air in your clipboard.

Pasting

Once you have something in your clipboard, you can paste it to world. The last argument is optional: if you want the copy to paste at the same point that it was copied at, type `//paste -o`, otherwise the paste will be placed relative to you. **Remember that if you are pasting relatively, it will be relative to where you were when you made the initial**

copy. For example, if you were on top of your castle when you copied it, pasting it would result in the castle being pasted under you.

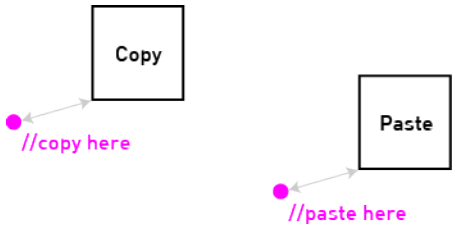


Fig. 4: A primer on how relative positions work for clipboards

Like the copy/cut commands, the paste command also allows the same three flags:

- `-e` can be specified to also paste entities, if your clipboard contains any
- `-b` can be specified to also paste biomes, if your clipboard contains any
- `-m <mask>` can be used to specify a *mask* of blocks to paste. Blocks that do not match the mask will not be pasted.

In addition, there are some additional flags:

- `-a` will not paste air from your clipboard. This is the same as `-m #existing`. The `-a` and `-m` flag *can* be combined (or you can just add `#existing` to your mask).
- `-s` will set your selection to the area you are pasting into.
- `-n` will set your selection like `-s` does, but will *not* actually paste anything. This can be useful to check where your clipboard will end up before actually pasting.
- `-o` will paste the clipboard back to its original origin, as explained above. This will disregard the entire "relative positions".

Rotating

Sometimes you may want to rotate your copy. The `//rotate <y> [x] [z]` command currently lets you rotate your copy around the Y (up-down) axis 90 degrees or at any multiple of 90 degrees. To be accurate, it actually allows you to revolve your copy around the relative offset that you were at when you originally made the copy. If you wanted to rotate a copy around its center, you would have had to stand in the middle of the copy when you had made it.

Note that the rotate command can also take an angle to rotate around the X or Z axis, though you must specify 0 for the axes that you don't use, e.g. for X axis rotation `//rotate 0 90`, and for Z axis `//rotate 0 0 90`. These two axes can be used to make something vertical, horizontal, or vice versa.

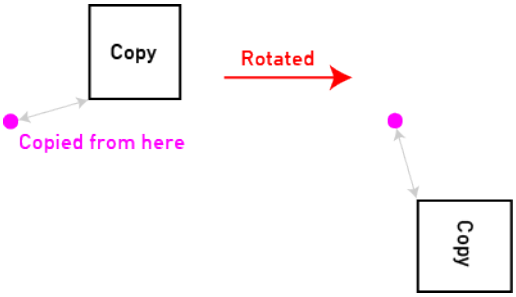


Fig. 5: Rotating around your relative position

Flipping

The `//flip [direction]` command flips the current clipboard across the plane perpendicular to the given direction. By default this direction will be whichever way you are facing, but you can also specify it explicitly. There are three planes you can flip across: XY, YZ, and XZ. The mapping used is included below for reference.

Directions	Plane
`north` or `south`	XY
`east` or `west`	YZ
`up` or `down`	XZ

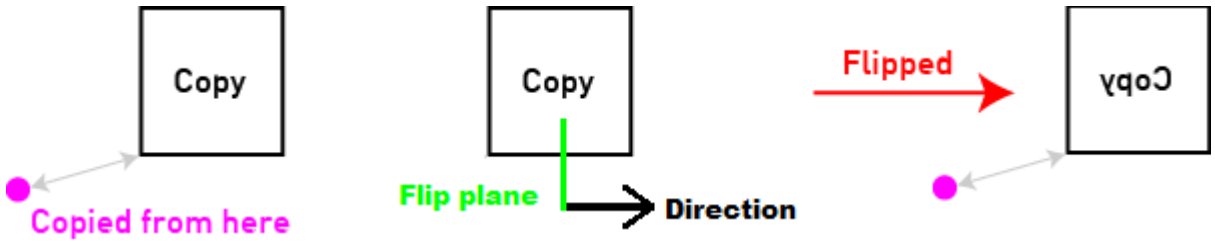


Fig. 6: Flipping the clipboard across a plane

Loading and Saving

WorldEdit can work with "schematic" files to save or load your clipboard to disk.

To save your current clipboard to file, use `//schem save <filename>`.

To load a saved schematic, use `//schem load <filename>`.

A note on schematic formats

Before WorldEdit version 7 (corresponding to Minecraft 1.13), the files were saved with a ".schematic" file extension in a format that was compatible with many other software such as MCEdit, Redstone Simulator, and more. Unfortunately, the format wasn't suited for the new block format Mojang was migrating to, so a new format was devised - named the Sponge schematic format, using the extension ".schem".

Note that WorldEdit can still import old ".schematic" files saved in older versions (or third party programs) through a legacy compatibility layer, but they can no longer be written to.

Relative positions and schematics

Both the origin of the copy and your offset to the copy are saved with the file so that you can load it back later on and paste the copy at its original location or relative to you as if you had copied it. You should be familiar with how `//copy` and `//paste` store your relative position.

Note that third party software which uses the format may not necessarily use relative positions as WorldEdit does, so they may not have that information.

Schematic Management

List available schematics

```
//schem list [-dn] [-p <page>]
```

The `-d` *or* `-n` flag may be used to sort by newest/oldest file modification time. The `-p` flag will get a specific page number. Note that the output of this command is interactive - the arrows at the bottom will retrieve the previous/next page automatically, and the `[L]` "button" on the left will load the schematic.

Deleting schematics

```
//schem delete <filename>
```

Listing available formats

```
//schem listformats
```

Although the note above only mentions the older "MCEdit" schematic format and the newer "Sponge" schematic format, WorldEdit actually has no limit on how clipboards are stored. Third-party plugins can register new formats with WorldEdit for saving and/or loading.

Schematic Storage

Schematics are saved to and loaded from WorldEdit's schematic folder, which is named `schematics` by default, but can be changed in the *config*. The folder is not created until you save a schematic in-game. If you've downloaded a schematic somewhere and want to add it, you can make the folder manually. The folder needs to be inside WorldEdit's config folder, which is `plugins/WorldEdit` on Bukkit/Spigot/Paper, and `config/worldedit` on other platforms. This means that **by default** the schematics folder is located at `plugins/WorldEdit/schematics` or `config/worldedit/schematics`.

Note

If you want to share schematic folders between servers/installations, or simply want to store them elsewhere, you will have to enable the "allow-symbolic-links" option in the config.

Tip

The save and load commands, although they ask for a file name, can take `folder/file`, in which case a sub-folder will be created in your schematics folder. This can be useful to organize your schematics.

Sharing

WorldEdit has an inbuilt way to easily share schematic files online.

To share your current clipboard online, use `//schem share`. You can also supply a name for the schematic, as well as an alternate upload destination and format using the `//schem share [name] [destination] [format]` syntax.

Share destinations

By default, WorldEdit uploads schematics to the EngineHub Paste Service. Other plugins and mods can use the WorldEdit API to register new share destinations that you can upload to via the destination argument of the command.

It's important to note, that schematics shared to the EngineHub Paste Service will be deleted after one month. This service should only be used for short-term share links, not as a long-term download location.

1.6.5 Generation

Sometimes you may want to generate forests or create spheres automatically, as doing it by hand may be too tedious. WorldEdit has a number of tools that allow you to do just that. These commands don't need a region; they use the block that you are standing in.

As most commands in WorldEdit, these commands all accept *patterns* as arguments.

Tip

These commands actually use your "placement position", which defaults to your current location. You can use the `//toggleplace` command to instead use your first selection point (the one selected via wand left-click, or `//pos1`).

Cylinders

```
//cyl <pattern> <radius> [height]
//hcyl <pattern> <radius> [height]
```

WorldEdit is capable of producing both hollow and filled cylinders as well as hollow and filled circles. It uses a fast algorithm to generate the objects and the algorithm is capable of creating nice and symmetrical edges. The cylinders are created at your feet and extend upwards. If you are creating a circle, you simply only need to create a cylinder of height 1.

Tip

For better control over the edges of the shape, you can use decimal numbers for the radius.

For elliptical cylinders, you can instead specify two radii: one for the east-west axis, one for the north-south axis.

```
//cyl <pattern> <radiusEW>,<radiusNS> [height]
//hcyl <pattern> <radiusEW>,<radiusNS> [height]
```

Example: Creating cylinders and circles

Creating a filled glass cylinder of radius 5 and height 10:

```
//cyl glass 5 10
```

Creating an elliptical hollow cylinder out of stone with radii of 5.5 and 15 and height of 1:

```
//hcyl stone 5.5,15 1
```

Sphere

```
//sphere [-r] <pattern> <radius>
//hsphere [-r] <pattern> <radius>
```

Both hollow and filled spheres can be created. By default, the center of the sphere will be the block above the one that you are standing on. If you provide the *-r* (raised) option, the sphere will be raised by its radius so that its bottom is at your feet instead.

Like cylinders, you can create ellipsoids by specifying multiple radii (which can again be decimals). The order of the radii is north-south axis, up-down axis, and then east-west axis.

```
//sphere [-r] <pattern> <radius>,<radius>,<radius>
//hsphere [-r] <pattern> <radius>,<radius>,<radius>
```

Pyramids

```
//pyramid <pattern> <size>
//hpyramid <pattern> <size>
```

This command creates a pyramid out of the given pattern with the specified *height*. That is, if you specify a height of say, 5, there will be 5 layers, each one block shorter than the last (in each direction, from bottom to top). The side-length of the lowest layer will be twice the height.

Forests

```
/forestgen [size] [type] [density]
```

Forests can be generated with this command. The size parameter indicates the width and height of the square area to generate the forests in. The density can vary between 0 and 100, and numbers like 0.1 will work. Note that the default density (of 5) is already rather dense. The size is the apothem (radius) of a cuboid, centered on your placement position.

> **Tip**
>
> This command operates the same as the forest generator in *region operations* except this one takes a size from your placement position, instead of using your selection.

> **Warning**
>
> Depending on the platform used, WorldEdit may not be able to //undo the generation of these trees.

Pumpkin patches

```
/pumpkins [size]
```

WorldEdit can generate some pumpkin patches. The size parameter is the width and height of the square area to generate the patches within, radiating out from your feet. The density of the patches is currently not adjustable.

Generating Arbitrary Shapes

```
//generate <pattern> <expression>
```

Aliases: `//g`, `//gen`

Generates any shape that can be described with a mathematical formula:

- A torus
- Rotated cylinders
- Jagged canyons
- Any shape you can imagine and boil down into a formula

This uses the *expression parser*.

Flags

- `-r` - Use raw coordinates, with one block equaling one unit
- `-c` - Shift the origin to the center of your selection, with one block equaling one unit
- `-o` - Shift the origin to your placement position (your position or pos1, with `/togglepos`), with one block equaling one unit
- Without any of these flags, coordinates will be normalized to -1..1 (from selection min/max points, meaning the entire selection is 2x2x2 units), note that each axis may be a different number of blocks per unit depending on your selection skewness.
- `-h` - Generate a hollow shape. Blocks will only be set if they neighbour any blocks that are not part of the shape.

Variables

- `x`, `y`, `z` (input) - Coordinates
- `type`, `data` (input/output) - Material to use, defaults to the block/pattern entered

Note

Since the expression parser only takes numbers as variables, type/data variables and query functions only work with blocks that have legacy type/data values. If you need to use it with newer blocks (> MC 1.13), use a placeholder and `//replace` that placeholder after generating your shape. The `<pattern>` arg of the command is not restricted, only the expression.

The expression should return true (`> 0`) for blocks that are part of the shape and false (`<= 0`) for blocks not part of the shape. The expression is tested for each block in your selection.

Shape Examples

Example: Generating various shapes

Torus of major radius 0.75 and minor radius 0.25:

```
//g stone (0.75-sqrt(x^2+y^2))^2+z^2 < 0.25^2
```

Gnarled hollow tree:

```
//g -h oak_log (0.5+sin(atan2(x,z)*8)*0.2)*(sqrt(x*x+z*z)/0.5)^(-2)-1.2 < y
```

Rainbow Torus:

```
//g white_wool data=(32+15/2/pi*atan2(x,y))%16; (0.75-sqrt(x^2+y^2))^2+z^2 < 0.25^2
```

Rainbow Egg:

```
//g white_wool data=(32+y*16+1)%16; y^2/9+x^2/6*(1/(1-0.4*y))+z^2/6*(1/(1-0.4*y))<0.08
```

A heart:

```
//g red_wool (z/2)^2+x^2+(5*y/4-sqrt(abs(x)))^2<0.6
```

Sine wave:

```
//g -h glass sin(x*5)/2<y
```

Radial cosine wave:

```
//g -h glass cos(sqrt(x^2+z^2)*5)/2<y
```

Circular hyperboloid:

```
//g stone -(z^2/12)+(y^2/4)-(x^2/12)>-0.03
```

Tip

Want more cool shapes? Try out a program like MathMod which comes with tons of shapes and helps you make more. Note that WorldEdit uses isometric (x,y,z) formulas, not parametric (u,v,t). Also, you may have to scale your x, y, and z variable depending on your selection size and the domain of the function.

Generating Biome Shapes

Just like the generate command, you can use an expression to set a biome in a particular shape. This uses the same syntax as above, but takes a biome id instead of a pattern.

Note

As of Minecraft 1.15, biomes are stored in 3 dimensions. Since 1.16, WorldEdit uses the full 3D biomes, but there are some new limitations from Mojang. See *Setting Biomes* for more details.

1.6.6 Tools

Sometimes, running commands over and over is too tedious for some tasks. WorldEdit comes with a wide variety of tools which can be bound to an item and will activate upon clicking on a block. Most tools activate on right-click, while some also have actions for left-clicks. Some tools, including all "brush"-style tools, can be used at a distance, and will act as if you clicked the block you are looking at, even from far away.

To use a tool, hold an item in your hand, then bind the desired tool by name using `/tool [name]` . You will get a message that the tool was bound to that item. Now, every time you activate the tool (by clicking with the item in your hand), that tool will perform its action. **To unbind a tool, hold the item and use the `/tool none` command.**

Tip

The selection wand (default: wooden axe, bound with `/tool selwand`) and navigation wand (default: compass, bound with `/tool navwand`) are technically tools. They are described on the *selections* and *navigation* pages respectively. You can bind and unbind them just as any other tool.

Tool Listing

Tree generation tool

```
/tool tree [type]
```

This tool will generate a tree of the chosen type when you right click a block. Note that it uses Minecraft's tree generator, and has the same limitations - it will not generate trees on unplantable blocks, or through solid blocks above (unless the tree would normally be able to grow there).

Floating tree remover

```
/tool deltree
```

Have players who chop down trees half-way and leave floating tree tops everywhere? This tool, upon right-clicking a floating leaf or log block (or mushroom block), will remove all connected floating tree blocks. This tool will not operate on trees that are still connected to something (such as the ground).

Block replacer tool

```
/tool repl <pattern>
```

This tool will replace the right-clicked block with a block from your pattern. You can also left-click a block to replace your current pattern with the left-clicked block.

Long range building tool

```
/tool lrbuild <leftclick pattern> <rightclick pattern>
```

This tool allows you to place and destroy blocks at a distance. Just aim and click. Blocks are placed as if you right clicked the block. If you set one of the blocks to air, it will instead delete the block you are targeting.

Long range wand

```
/tool farwand
```

This tool works just like the normal selection wand - but at any distance. Instead of being `//pos1` and `//pos2`, it's `//hpos1` and `//hpos2`.

Cycler Tool

```
/tool cycler
```

This tool can be used to cycle a block's states. Left-clicking will choose which property to cycle, and right-clicking will cycle the available values of that property. This is useful for example, to rotate individual blocks in place.

Query Tool

```
/tool info
```

The query tool will show information about the right-clicked block. It will show the coordinates, the block type, states, light level (emitted/above), and the internal id (if available).

Flood fill tool

```
/tool floodfill <pattern> <range>
```

The flood fill tool will, starting at the right-clicked block, change it and all connected blocks of the same type (within the given range) to the specified pattern.

Super-pickaxes

Super-pickaxes are slightly different than other tools. Instead of being bound to a single item, they are just toggled on or off with their commands. When on, left-clicking with *any* pickaxe in your hand will trigger the superpick. Unlike normal tools, they are toggled off with `//`.

Single Superpick

```
//
/sp single
```

This super-pickaxe allows you to instantly break blocks. That may seem redundant, but it predates creative mode (creative mode was removed before alpha and wasn't re-added until Beta 1.8). It also will drop the blocks you break as items (*configurably*), which creative mode does not do.

Area Superpick

```
/sp area <radius>
```

This super-pickaxe will break all blocks matching the same type as the initially clicked block within the radius. Like the single mode superpick, it can be configured to drop items.

Recursive Superpick

```
/sp recur <radius>
```

This super-pickaxe also breaks all blocks matching the same type as the initially clicked block within the radius, but only those that are connected (via matching blocks) to the original. Think of this as a "vein miner" mode.

Brushes

Unlike the above tools, which are mostly for utility, another group of tools, known as brushes, are mainly designed for painting and building quickly from afar.

They can be found on the *brush page*.

1.6.7 Brushes

Brush tools are general more designed for building, sculpting, and painting than the general utility *tools*. Like the rest of the tools, they are bound to an item by using the command, and are activated by right-clicking (or left clicking, for those with two actions). They are unbound with the `/brush none` command.

Brushes have a few unique settings available to them. Brushes allow you to choose a mask, size, pattern, and range. These allow fine-tuning how you build and paint.

Brush Listing

Sphere brush

```
/brush sphere [-h] <pattern> [radius]
/br s [-h] <pattern> [radius]
```

The sphere brush, as its name suggests, creates sphere at the target point. The `-h` flag will make a hollow sphere.

Cylinder brush

```
/brush cylinder [-h] <pattern> [radius] [height]
/br cyl [-h] <pattern> [radius] [height]
```

The cylinder brush creates cylinders of the given radius and height. `-h` is hollow as usual.

Set brush

```
/brush set <shape> [size] <pattern>
```

The set brush can set spheres, cylinders, or cuboids of the given size and pattern. It is mostly redundant to the previous two brushes, unless you need a cube.

Clipboard brush

```
/brush clipboard [-aoeb] [-m <mask>]
```

The clipboard brush uses your clipboard as a shape and places it at the target location each brush activation. The `aebm` flags all work just like the *paste command*. The `-o` flag is a bit different - it makes the clipboard paste at the brush's location as the origin. Otherwise, the clipboard is centered to that location. Choose your relative position carefully when copying a clipboard for use with this brush.

Smooth brush

```
/brush smooth [radius] [iterations] [mask]
```

Smoothes the area, just like the `//smooth` command explained in *region ops*. You can specify the mask of blocks to consider while building a heightmap (this is separate from the mask of blocks that will be affected).

Gravity brush

```
/brush gravity [radius] [-h]
```

The gravity brush will move blocks downwards within the affected brush area. Using the `-h` flag will start from the max y, instead of from the top of the brush area.

Forest brush

```
/brush forest <shape> [radius] [density] <tree type>
```

Like the `//forest` and `//forestgen` commands, this brush plants trees, using the shape, radius, density, and tree type provided.

Extinguish brush

```
/brush extinguish [radius]
/br ex [radius]
```

A handy shortcut for a sphere brush of air masked to fire blocks, in case something is burning and you don't have time to type in 3 commands to put it out.

Butcher brush

```
/brush butcher [radius] [-pngabtfr]
/br kill [radius] [-pngabtfr]
```

Just like the `//butcher` *utility command*, the butcher brush kills entities in its area. Note that the radius is strictly cylindrical, but goes from minimum to maximum world height. It is not a sphere. By default, it only kills hostile mobs. The flags can be specified to determine what other mobs will be butchered.

Butcher flags

Flag	Description
`-p`	Kills tamed pets
`-n`	Kills NPCs
`-g`	Kills golems
`-a`	Kills animals
`-b`	Kills ambient mobs
`-t`	Kills mobs with name tags
`-f`	Combines all of the above flags
`-r`	Kills armor stands

Deform brush

```
/brush deform <shape> [size] [expression] [-ro]
```

Just like the `//deform` command described in *region operations*, this brush will apply an expression to blocks within the area given by the shape and radius. Also just like the command, `-r` uses raw coordinates, and `-o` offsets from your placement position. The default is the brush target point.

Raise Brush

```
/brush raise <shape> [size]
```

A special case of the deform brush which uses the expression `y-=1`.

Lower Brush

```
/brush lower <shape> [size]
```

A special case of the deform brush which uses the expression `y+=1`.

Biome brush

```
/brush biome <shape> [radius] <biomeType>
```

Sets the biome within the area given by the shape and radius. Keep in mind that since 1.15 the biomes are 3D and the smallest area you can change is a 4x4x4 cuboid. Effects can't be seen until you rejoin the world.

Brush Settings

These commands modify the settings on your *currently selected* brush only. Each brush you have bound has its own settings.

Not all settings are used by all brushes - the clipboard brush doesn't have a size setting (it uses your clipboard's size), the butcher brush doesn't have a mask or material (it affects entities, not blocks), and so on.

Mask

```
/mask [mask]
/mask
```

Sets a *mask* on your current brush, which restricts what blocks will be affected by it. Not specifying a mask will disable it for your brush, allowing it to affect everything again. Note that if you already have a global mask set with `//gmask`, it will be combined with this one.

Size

```
/size [size]
```

Sets the size of the brush. Generally, this means the radius or affected area. Note that the maximum size is *configurable*.

Material

```
/material <pattern>
```

Sets the pattern used by the brush.

Range

```
/range <range>
```

Sets the maximum range that the brush will try to build at. Note that with a short enough range, brushes *will build in mid-air* if they can't find a block in your ray trace.

Trace Mask

```
/tracemask [mask]
/tracemask
```

Sets the mask used for the ray tracer. By default, brushes will perform their action on the first non-air block (or when the trace hits the range, whichever comes first). By setting the tracemask, you can make brushes trace through any block you choose. For example, `/tracemask #solid` will go through non-solid blocks, such as water. This is useful for building underwater, through walls, and whatever else.

1.6.8 Utilities

WorldEdit provides a handful of utility commands. These are often conveniences for common tasks which would require multiple commands and selections otherwise. All these commands operate on your placement position. By default, this is your player position, but it can be changed to your first selection position using the `/toggleplace` command.

- *Fixing pools*
- *Simulating snowfall*
- *Thawing snow*
- *Simulating grass growth*
- *Extinguishing fires*
- *Removing mobs*

Editing nearby blocks

Sometimes you want to set or replace blocks in an area, but don't need a precise selection - you just want to use an area around you.

Removing blocks above and below:

```
/removeabove <size> [height]
/removebelow <size> [depth]
```

These two commands let you easily remove blocks above or below you. An example usage is to remove those tower blocks people create in order to get to a high point. The size parameter indicates the size of the cuboid to remove. The cuboid's width and length will be `(size - 1) * 2 + 1`. The center of the cuboid is the block above the one that you are standing on. If you don't specify a height or depth, the commands will extend to the extents of the world.

Removing nearby blocks:

```
/removenear <mask> <size>
```

This command removes nearby blocks of a certain type. The size parameter indicates the size of the cuboid to remove. The cuboid's width and length will be `(size - 1) * 2 + 1`. The center of the cuboid is the block above the one that you are standing on.

Replacing nearby blocks:

```
/replacenear <size> <mask> <pattern>
```

If you need to quickly replace nearby blocks, this command is a nice shortcut. The size parameter indicates the size of the cuboid to replace. The cuboid's width and length will be `(size - 1) * 2 + 1`. The center of the cuboid is the block above the one that you are standing on.

Filling pits

```
//fill <pattern> <radius> [depth]
```

The fill command will start at the your placement position and work outward and downward, filling air with the given pattern. Note that it only works straight downward from the starting layer, so while it will fill a pond, it will not fill a cave (that goes "outward" as it goes down).

The fill command will never go upwards from your starting position.

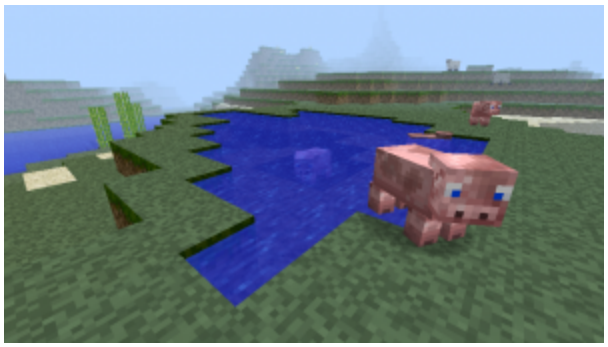


Fig. 7: An irregularly shaped pool filled with `//fill`. You wouldn't have been able to replace air with water (with `/replace`) in this situation because the area doesn't fit neatly into a cuboid.

Recursive Fill

```
//fillr <pattern> <radius>
```

Unlike the fill command, the recursive fill command *will* work outwards as it moves down, meaning it can fill caves and other holes that expand into the walls.

Like the fill command, it will not travel upwards beyond the starting y level.

Draining pools

```
//drain [-w] <radius>
```

The drain command can empty a pool of water or lava. It only removes connected blocks from the starting position, so it will not drain non-connected pools even if they are in the radius.

Adding the `-w` flag will also un-waterlog blocks, leaving them dry.

> **Note**
>
> Some blocks, like kelp and seagrass, despite looking like they are waterlogged, do not actually have a "dry" state. If you're trying to drain an ocean or river with these blocks, use `//removenear` (explained above) to remove those blocks first.

Fixing pools

```
/fixwater <radius>
/fixlava <radius>
```

This command will replace "flowing" versions of lava and water with "stationary" ones within the radius. This was useful for fixing water features that had been partially removed with buckets or blocks broken. Note that since Mojang changed water mechanics relatively recently, `fixwater` is much less useful than before.

Simulating snowfall

```
/snow <radius>
```

Cover snow over the general area! This algorithm will only cover blocks with snow if they should be covered (for example, torch blocks will not be covered). If an area has something above it (like an overhang), snow will not reach it. "Snowfall" is completely vertical.

> **Note**
>
> The snow command does not yet build up layers of snow.

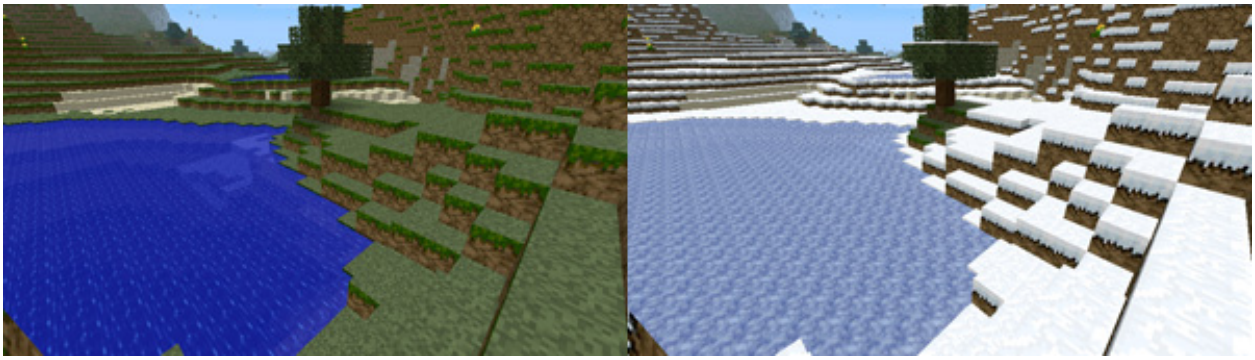


Fig. 8: Snowfall transforming a landscape

Thawing snow

```
/thaw <radius>
```

This command works exactly in the opposite way as the `/snow` command above, removing snow and ice from exposed areas.

Simulating grass growth

```
/green [-f] <radius>
```

Cover grass over the general area! This area works horizontally outwards to convert dirt into grass.

Using the `-f` flag will also turn coarse dirt into grass as well as regular dirt.

Extinguishing fires

```
/ex [radius]
```

This is essentially a shortcut `/removenear fire <radius>`, to allow you to quickly put out fires.

Removing mobs

```
/butcher [-pngabtfl] [radius]
```

This command removes nearby mobs. If you don't specify a radius, all active mobs in the entire loaded world will be removed. The mobs will not drop their loot. Be aware that even if you kill all mobs, they will come back quickly if there isn't anything else preventing spawning.

Butcher flags

Flag	Description
`-p`	Kills tamed pets
`-n`	Kills NPCs
`-g`	Kills golems
`-a`	Kills animals
`-b`	Kills ambient mobs
`-t`	Kills mobs with name tags
`-f`	Combines all of the above flags
`-r`	Kills armor stands

1.6.9 Snapshots

A very powerful feature of WorldEdit is that it can load a section of your world, defined by your *selection region*, and restore it from a backup without having to shutdown your server or use an offline map editor. A large number of problems can be easily fixed this way, from undoing a griefer's work to fixing a world save error to even rolling back from a project you gave up with half-way through.

Configuring Snapshots

In order for WorldEdit to be able to read your backups, you will have to choose a directory to place backups inside. The path can be set in the *configuration* and is relative to the server/`.minecraft` root folder (not the WorldEdit folder!). You can also use an absolute path if your backups are stored outside your server folder (like on another disk, which is recommended in case of hardware failures).

Once set, just toss either copies of your world folder or zipped copies of your world folder into your backup folder. An example of how you could lay out your backups folder is below.

Example: Possible structures of backup storage


`backups/` (this is the folder name in the config)


`world/` (this is the name of a world)

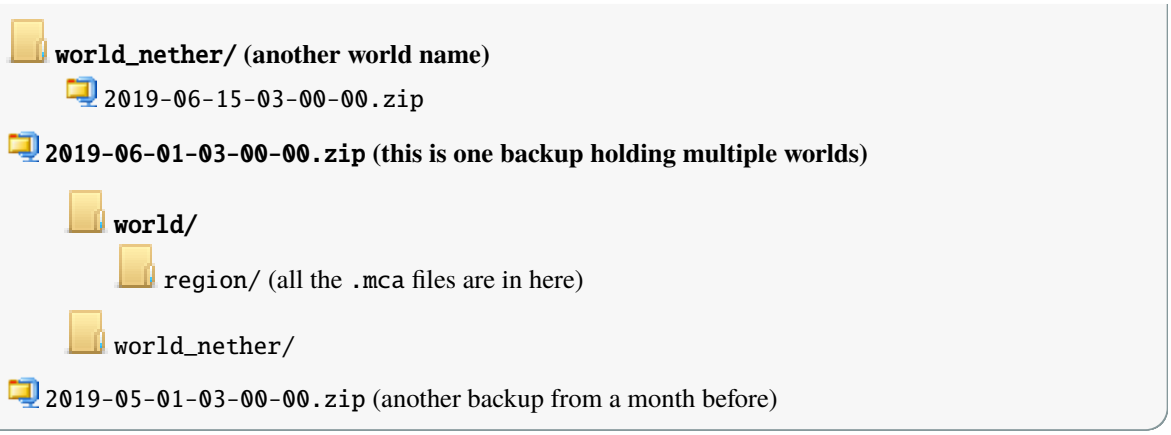

`2019-06-15-03-00-00.zip` (a backup for just this world)


`world/` (must match the world name above)


`region/` (contains `.mca` files for this world)


`2019-05-15-03-00-00` (backups don't have to be zipped)


`world/`



As you might have noticed, each individual backup must have a timestamp. WorldEdit expects these timestamps in order to determine which backups are the newest. The world folder must be inside the backup, with a region folder inside that world folder. You can either have backups at the top level with multiple world folders inside, or multiple world folders with backups inside for each individual world.

Tip

If you use a Linux-like system, you can use the following line to create a world backup ZIP having an acceptable filename: `zip -v backups/`date "+%Y-%m-%d-%H-%M-%S"`.zip -r world`.

Supported Archive Formats

WorldEdit natively supports ZIP files, using Java's zip library. However, Java's zip support only supports basic zip files. If you receive cryptic errors while using zip files, you may want to install TrueZip. If you want to use another archive format, such as tarballs, TrueZip will also support these.

TrueZip can be installed by downloading the JAR file from the maven repository and saving it as `truezip.jar`. The jar should be placed in the `plugins` or `plugins/WorldEdit` folder on Bukkit, or in the `mods` folder on other platforms.

Tip

Using backup archives (e.g. zip files) will save disk space at the cost of increasing CPU when actually restoring. The trade-off is up to you to decide.

Using Snapshots

Now that your snapshots are configured, using them is a breeze. Just *select an area*, and use `//restore`.

By default, WorldEdit will find the most recent backup for your current world, and restore your selected area from it.

If you don't want to use the most recent snapshot (perhaps the damage was already done and you need an older one), there are additional commands to choose which snapshot to use.

To get started, use `/snap list`. This will list all snapshots available for your current world.

You can either use `/snap use latest` or `/snap use [name]` to select either the latest snapshot, or by name. You can also use `/snap sel <number>` to use the one with that number in the list.

If you know a time point which you either need a backup before or after that point, you can use `/snap before <time>` or `/snap after <time>` to find the closest snapshot before/after the given time. These commands take a timestamp like the file names, or even a natural time, such as `/snap before "last friday"`.

1.6.10 Other

Expression Syntax

The WorldEdit expression parser is supposed to work like Java and related languages, with a few subtle differences:

- Final semicolons can be omitted in most cases.
- The last value in a sequence is always returned, even without a return statement.
- The binary infix ^ operator is a power operator instead of an xor operator and has an according priority as well.
- There is a postfix factorial operator (!).
- There is a binary infix near operator (~=).
- No objects :)

Operators

The expression parser uses Java's precedence rules, with the following exceptions and additions:

- The binary power operator (^) is between priority 2 and 3
- The postfix factorial operator (!) has a priority of 2
- The near operator (~=) has a priority of 7

Binary infix

These operators are put between their two operands.

Arithmetic

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder
^	Power

Bitwise

These operators interpret their operands as 32 bit integers and operate on their bits.

<<	Left-shift
>>	Right-shift

Logical

These operators interpret everything greater than zero as true and everything else as false. They return 1 for true and 0 for false.

&&	Logical and
||	Logical or

Comparison

These operators compare their operands and return 1 for true and 0 for false.

<	less than
>	greater than
<=	less or equal
>=	greater or equal
==	equal
!=	not equal
~=	near

Assignment

These operators require a variable on the left side. Using the simple assignment operator (=) to assign to a non-existant variable creates a temporary variable.

=	simple assignment
+=	addition+assignment
-=	subtraction+assignment
*=	multiplication+assignment
/=	division+assignment
%=	remainder+assignment
^=	power+assignment

Prefix

These operators precede the expression they apply to.

Prefix Operators

-x	(negation)
~x	Bitwise complement (see bitwise binary operators)
!x	Logical complement (see logical binary operators)
++x	Pre-increment
--x	Pre-decrement

Postfix

These operators succeed the expression they apply to.

Postfix Operators

x!	Factorial
x++	Post-increment
x--	Post-decrement

Ternary infix

The ternary operator is used to represent a conditional expression in a compact way:

```
<condition> ? <true-branch> : <false-branch>
```

It works exactly like the if/else statement, except that the branches can only be single expressions.

Functions

Math Functions

The expression parser provides the following functions from the Java Math library:

`abs`	Returns the absolute value of a number.
`acos`	Returns the arc cosine of a value; the returned angle is in the range 0.0 through pi.
`asin`	Returns the arc sine of a value; the returned angle is in the range -pi/2 through pi/2.
`atan2`	Returns the angle theta from the conversion of rectangular coordinates (x, y) to polar coordinates (r, theta).
`atan`	Returns the arc tangent of a value; the returned angle is in the range -pi/2 through pi/2.
`cbrt`	Returns the cube root of a value.
`ceil`	Returns the smallest (closest to negative infinity) value that is greater than or equal to the argument and is equal to a mathematical integer.
`cos`	Returns the trigonometric cosine of an angle.
`cosh`	Returns the hyperbolic cosine of a value.
`exp`	Returns Euler's number e raised to the power of a value.
`floor`	Returns the largest (closest to positive infinity) value that is less than or equal to the argument and is equal to a mathematical integer.
`ln`	Returns the natural logarithm (base e) of a value.
`log`	Returns the natural logarithm (base e) of a value.
`log10`	Returns the base 10 logarithm (base e) of a value.
`max`	Returns the greatest of the values. (supports 2 and 3 arguments)
`min`	Returns the smallest of the values. (supports 2 and 3 arguments)
`rint`	Returns the number that is closest in value to the argument and is equal to a mathematical integer.
`round`	Returns the closest number to the argument
`sin`	Returns the trigonometric sine of an angle.
`sinh`	Returns the hyperbolic sine of a value.
`sqrt`	Returns the correctly rounded positive square root of a value.
`tan`	Returns the trigonometric tangent of an angle.
`tanh`	Returns the hyperbolic tangent of a value.

Additional Functions

Additionally, it provides the following functions:

`rotate(x, y, angle)`	Rotates the given coordinate pair by the given angle, in radians.
`swap(x, y)`	Swaps the contents of the 2 given variables.
`random()`	Returns a random positive number less than 1.0.
`randint(max)`	Returns a random positive integer less than max.
`perlin(seed, x, y, z, frequency, octaves, persisence)`	Generates perlin noise with the given parameters.
`voronoi(seed, x, y, z, frequency)`	Generates voronoi noise with the given parameters.
`ridgedmulti(seed, x, y, z, frequency, octaves)`	Generated ridged multi fractal noise with the given parameters.

Block Query Functions

The following functions can be used to query blocks in a world in an editing context. Note that they still use legacy ID and data, so they may have undefined behaviors for new (1.13+) blocks.

`query(x, y, z, type, data)`	Returns true if the block at the given coordinates has the given legacy id and data value. If type or data are variables, the id and data of the block will be assigned to that variable.
`queryRel(dx, dy, dz, type, data)`	Like query, except with an offset from the currently evaluated block coordinates
`queryAbs(xp, yp, zp, type, data)`	Like query, except with absolute world coordinates

Buffer Functions

These functions provide access to data buffers (essentially, arrays). Two buffers are provided, one is a global shared buffer, and one is local to the expression. The function with *g* prepended accesses the global buffer, without the *g* accesses the local buffer.

`(g)megabuf(index)`	Returns the value of the buffer at the given index.
`(g)megabuf(index, value)`	Sets the value of the buffer at the given index.
`(g)closest(x, y, z, index, count, stride)`	Finds the index of the closest set of x,y,z values (as in, three consecutive buffer values) to the given x,y,z values within *count* iterations and *stride* space between each iteration, starting at the given index value.

Constants

Constants

The following constants are always available, and cannot be assigned.

`e`	2.7182818284590452354	The base of the natural logarithm
`pi`	3.14159265358979323846	The ratio between circumference and diameter of a circle
`true`	1	for boolean operations
`false`	0	for boolean operations

Block Statements

Block statements are groups of statements enclosed in braces:

```
{ x=5; y=6; }
```

They are mostly used in conjunction with control structures.

Control Structures

if/else

```
if (<condition>) <true-branch>
if (<condition>) <true-branch> else <false-branch>
```

- <condition> is evaluated to decide which branch to execute.
- Everything greater than zero is interpreted as true and everything else as false.
- <true-code> and <false-code> can either be single statements delimited with a semicolon or block statements.

Note: An else keyword is always associated with the last if. This allows elseif constructs like these:

```
if (<condition 1>) <true-code 1> else if (<condition 2>) <true-code 2> else <false-
↪code>
```

Loops

Loops can at most loop 256 times.

while

```
while (<condition>) <body>
do <body> while (<condition>);
```

- <condition> is evaluated to decide whether to continue looping.
- <body> can either be a single statement delimited with a semicolon or a block statement.
- do-while checks the condition after executing the body.

Java/C-style for

```
for (<init>; <condition>; <increment>) <body>
```

- <init>, <condition> and <increment> are single expressions.
- <body> can either be a single statement delimited with a semicolon or a block statement.

Execution steps

First, <init> is evaluated once, then, each iteration follows these steps:

1. If <condition> evaluates as less than or equal to zero (i.e. false), the loop is aborted.
2. <body> is executed.
3. <increment> is executed.

Simple for

```
for (<counter> = <first>, <last>) <body>
```

- <counter> is a variable used to count the iterations.
- <first> and <last> are single expressions.
- <body> can either be a single statement delimited with a semicolon or a block statement.

Execution steps

First, an internal counter is set to <first>. Then, each iteration follows these steps:

1. If the internal counter exceeds <last>, the loop is aborted.
2. <counter> is set to the internal counter.
3. <body> is executed.
4. <counter> is incremented by 1.0.

<first> and <last> are only evaluated once.

CraftScripts

Scripts allow you to program small tasks without having to learn Java, figure out how to compile WorldEdit, or bother with reinventing the wheel. CraftScripts are written in JavaScript.

Requirements

Before you start using CraftScripts, you'll have to install the Rhino JavaScript engine. A direct link to the download is here. Rename the downloaded file to `js.jar`. Move `js.jar` to the `plugins/` or `plugins/WorldEdit` folder (on Bukkit) or the `mods` folder (other platforms).

Using CraftScripts

Once you have the JS engine installed, drop your CraftScript `.js` files in the `craftscripts` folder (in the WorldEdit config folder - either `plugins/WorldEdit` or `config/WorldEdit` depending on platform).

To run a CraftScript

```
/cs <filename> <args>
/.s <args>
```

The `/cs` command will run the CraftScript with the given filename (`.js` can be left out). Each CraftScript may have its own arguments. The `/.s` command will re-run your last used CraftScript.

Writing CraftScripts

Scripting in WorldEdit allows you to write world manipulation code without having to learn Java or compile your code. Scripts, called CraftScripts in WorldEdit, and are written in JavaScript and go into your craftscripts/ directory. The advantages of writing scripts with WorldEdit are:

- Hook right into WorldEdit's undo/redo system
- Use WorldEdit's block place prioritization
- Accept WorldEdit's powerful block type syntax (`//set sign[facing=north]`)
- Get the region selected by the user

Note

It is recommended you have a basic understanding of JavaScript or Java to begin writing CraftScripts.

Tip

While we'll be going over using CraftScripts with the WorldEdit API, there are no real restrictions on what you can do. Advanced users may even hook into the API of the underlying platform (Bukkit, NeoForge, etc).

Introduction

Scripts have the following three variables in their global namespace:

- `context` is an instance of CraftScriptContext.
- `player` is the player who invoked the script, an instance of Player.
- `argv` is a Java array strings, which are the arguments used upon invoking the scripts

Working with blocks

All block editing in WorldEdit is done through an EditSession. This object handles history and block placement order all automatically. To get an edit session for your own script, use:

```
var sess = context.remember();
```

Every time you call that method, you will get a new `EditSession`, so be sure to keep one around. To set blocks, you will either need to provide a `BlockState` which is a combination of a block type and one or more states, or a `BaseBlock`, which is a `BlockState` that may additionally have NBT data.

Example: Setting a block to white wool

```
importPackage(Packages.com.sk89q.worldedit.world.block);

var sess = context.remember();
sess.setBlock(player.getBlockOn().toVector().toBlockPoint(), BlockTypes.WHITE_WOOL.
↪getDefaultState());
```

Note that because `BlockTypes` is in the `com.sk89q.worldedit.world.block` namespace, it had to be imported first. The first argument for `setBlock()` is a `BlockVector3` indicating the position in the world.

To get blocks, use `getBlock()` on `EditSession`. You'll get back a `BaseBlock` too.

Processing arguments

Arguments are passed in under the `argv` variable. If you need to check whether the right number of arguments was provided by the user, you can use `CraftScriptContext#checkArgs()`.

The `CraftScriptContext` can to some basic argument parsing with `CraftScriptContext#getBlock()`. You can also hook directly into WorldEdit's parsers via `WorldEdit.getInstance().getPatternFactory()` and `.getMaskFactory()`.

Example: Checking arguments

```
context.checkArgs(1, 3, "<block> [width] [height]");
var block = context.getBlock(argv[1]);
```

What happens if the user inputs an invalid block? An exception will be raised and if you don't catch it, the user will be informed about their error and your script will be halted.

Working with Java packages

To import a java package, you can use the following syntax:

```
importPackage(Packages.package.name.here);
```

You can import any package available in the Java classpath - not restricted to WorldEdit.

Example Scripts

You can find some example scripts in the GitHub repository for WorldEdit. Note that they may not all be updated for current WorldEdit API. You can find more about the WorldEdit API in the *API section*.

1.7 Developer API

1.7.1 API Concepts

WorldEdit uses some simple concepts to create flexible operations. These concepts are detailed below, and should be read in the order provided, as later concepts sometimes reference earlier concepts.

Actors

An `Actor` is anything that uses WorldEdit commands or certain APIs. Typically, an actor is a player, but they can also be a command block, the console, or perhaps an entity. As should be clear, an actor does not need a location in the world, although many actors also implement `Locatable` to provide this. A mod/plugin does not usually need to provide an actor to use the WorldEdit API, but actors may be provided in certain hooks such as `EditSessionEvent` (see *Edit Session Event*), allowing for customization based on who/what an actor is.

Local Sessions

A `LocalSession` handles the *session data* of *actors*.

That means you'll need to access a player's LocalSession if you want to use their selection, clipboard, history, etc.

To retrieve a player's (or other actor's) session, the `WorldEdit` class provides access to the `SessionManager` via the `getSessionManager()` method. The `SessionManager` allows you to retrieve sessions via the `get(SessionOwner)` and `getIfPresent(SessionOwner)` methods - the latter may return null (for example if that player has logged off and their session expired).

Using LocalSession

Once you have a session object, the various methods on it will allow you to access the various session data mentioned above.

It is important to not make assumptions about the state or availability of various things held in a session. For example, the `getClipboard()` and `getSelection(World)` methods will throw exceptions if the user hasn't copied anything to their clipboard or hasn't made a selection in the provided world. If you want to require these, make sure you catch the exception and display a message to the user.

Regarding selections, note that WorldEdit allows selecting and editing cross-world (i.e. it is not bound to the world the player is in). You can check the current selection world via the `getSelectionWorld()` method. If you would like to limit the selection to the current world, pass the player's world to the `getSelection(World)` method.

Also note that a `RegionSelector` (`getRegionSelector(World)`) is not the same as a `Region`. If you only need read-only access to the player's selected region, stick to `getSelection(World)`. Selectors are only needed if you need to work with selection points, even if the region is not fully defined, or if you want to actually change the selection. Likewise, you should not mutate the `Region` object returned from `getSelection(World)` unless you

subsequently update the player's selector via `learnChanges()` and re-send necessary information to the player via `explainRegionAdjust(Actor player, LocalSession session)`.

For a concrete example of getting a player's selection, see the *examples page*.

The other common use of a `LocalSession` is accessing the player's *history*. If you perform some operation on behalf a player via an *edit session*, you can use the player's `LocalSession` to `remember` the `EditSession`, which will allow them to `//undo` that operation. You can also manually `undo` and `redo` via the `LocalSession`.

> **Note**
>
> History manipulation via `LocalSession` is only for changes caused by or attributed directly to a player (i.e. the session owner). If you store an `EditSession` outside of player history (e.g. for an automated operation that runs independently of player action), you can undo those changes directly via the `EditSession#undo(EditSession)` method.

Blocks

Blocks are broken into two parts, *type* and *state*. These are represented by the `BlockType` and `BlockState` classes. An example of block type is `minecraft:oak_log`, and a state would be the combination of the type `minecraft:oak_log` and the *properties* `[axis=y]`.

You can get a `BlockState` from a `BlockType` using either `getDefaultState()` or providing the correct property mappings to `getState(Map)`.

For example, to get the state for `minecraft:oak_log[axis=y]`

```
BlockType oakLog = Objects.requireNonNull(BlockTypes.OAK_LOG);
BlockState yFacingOakLog = oakLog.getState(ImmutableMap.of(
    oakLog.getProperty("axis"), "y"
));
System.err.println("State: " + yFacingOakLog);
```

Some blocks include NBT, and these are represented by the `BaseBlock` type.

You can get a `BaseBlock` from a `BlockState` using `toBaseBlock(CompoundTag)`.

Patterns and Masks

Patterns and masks are the same as described in *Patterns* and *Masks*. The only difference is that they must be constructed via their respective classes, rather than from formatted strings.

A single block pattern can be represented using a `BlockStateHolder`, such as `BlockState` and `BaseBlock`. Other patterns types are fairly obvious from their names, such as `TypeApplyingPattern` or `RandomStatePattern`. Use your IDE to find subclasses of `Pattern`.

Masks are a slightly different story. Exact and fuzzy block *state* masks are done using `BlockMask`, but you can also mask only over block *type* (`BlockTypeMask`) or only the *properties* (`BlockStateMask`). There are also some utility masks in the `Masks` class. Again, using your IDE to find `Mask` subclasses is recommended.

Extents

Extents form the backbone of WorldEdit's block manipulation. Extents are generally split into three categories: input, output, and both. Although extents provide and receive block and biome information, they are not always associated with a world / dimension.

Input extents are responsible for providing block and biome information for a given location. They do not provide a way to set blocks.

Output extents are responsible for receiving block and biome information for a given location. They do not provide a way to get blocks.

Most or all extents in WorldEdit implement both `*Extent` interfaces, typically through the `Extent` interface. `Extent` instances also provide a minimum and maximum point, as well as entity manipulation methods.

Some examples of extents are worlds and clipboards. Many block placement features in WorldEdit - such as fast and reorder mode - are implemented using `AbstractDelegateExtent` and hooking into `setBlock`.

Regions

WorldEdit uses regions to define where an operation works. A `Region` is a set of positions, potentially associated with a `World`. There is no requirement that a region be fully continuous.

Regions implement `Iterable<BlockVector3>`, meaning that the best way to get all the points of *any* region is to use a `for-each` loop, e.g. `for (BlockVector3 point : region)`. In addition to this, there are convenience methods to retrieve the minimum, maximum, center, area, width, height, length, chunks, and cubic chunks. You can also easily `expand()`, `contract()`, and `shift()` regions, which work like the commands of the same name. Note that these methods *do not* change any blocks.

Creating a region is as simple as calling the appropriate constructor of the region you want. Typically you want a `CuboidRegion`, but there are other subclasses for each of the selection types in WorldEdit, or you can implement your own!

Some common region usages are the actor's selection and clipboard bounds.

Registries

Almost everything in Minecraft uses the same format for identifying a particular type, such as `minecraft:stone`. This is known as a namespaced ID (see page for details). WorldEdit keeps some *registries* that allow access to blocks, items, biomes, entities, fluids, and more from the current Minecraft platform using their ID in a platform-independent way.

These registries are available on most classes named `*Type`, such as `BlockType` and `ItemType`. However, it is recommended to use the `*Types` classes instead, which either provide potentially-null constants or a `get` method for retrieving types that aren't built-in, such as modded blocks.

> **Note**
>
> The constants on these registries may be `null` because the API does not change across Minecraft versions, and this means that some blocks won't exist on earlier Minecraft versions, or like in 1.13, were renamed from their ID in 1.12 and WorldEdit no longer recognizes the block properly. If you require a block, you should wrap the access to the constant in `Objects.requireNonNull` or a similar check to properly communicate that you do not expect the constant to be missing at runtime.

Edit Sessions

An `EditSession` handles the configuration for getting and placing blocks, entities, and biomes. It sets up the chain of extents to place blocks properly. It also handles turning on and off reorder mode, fast mode, and buffering. Every operation targeting a world will use an `EditSession` at some point to ensure that block placement is done properly and quickly.

The easiest way to make a simple `EditSession` is using the helper methods on the `WorldEdit` class, `newEditSession(World)` or `newEditSession(Actor & Locatable)`. These are shorthands for using the full builder, available from `newEditSessionBuilder()`. The builder has all of the options that used to be available from the `EditSessionFactory`'s various methods, but in a easier-to-read form. If you use the builder, don't forget to call `build()` at the end to get an actual `EditSession`.

```
// Getting an edit session for a WorldEdit World
WorldEdit.getInstance().newEditSession(world)

// Getting an edit session for a WorldEdit World with a maximum block count
WorldEdit.getInstance().newEditSessionBuilder().world(world).maxBlocks(1000).build()
```

Edit sessions must be closed once all operations are complete, to ensure that block queues are flushed. They are not re-usable for edits after being closed, and a new one must be created for subsequent operations.

A clean way to close a `EditSession` is to use the `try-with-resources` statement:

```
try (EditSession editSession = WorldEdit.getInstance().newEditSession(world)) {
    // use the edit session here ...
} // it is automatically closed/flushed when the code exits the block
```

For more information, see The try-with-resources Statement page from the Java Tutorials.

Once an edit session has been used for editing and is closed, it stores all the changes performed. If you want to undo these changes at a later time, you can hold a reference to the edit session. Then, to undo the changes, make a new EditSession as detailed above, and call `editSession.undo(newEditSession)`.

The Extent Stack

The "extent stack" is a term used to describe the many extents that are composed to create an `EditSession`. At the very bottom of the stack is the `World` that the changes will be committed to. That extent is wrapped in other extents that all apply various fixes for oddities with Minecraft, and then the first event is posted with a stage of `Stage.BEFORE_CHANGE`. The resulting extent is used for `EditSession.rawSetBlock()`. Then, that extent is wrapped in some reordering / batching extents, and another event is posted with a stage of `Stage.BEFORE_REORDER`. The resulting extent is used for `EditSession.smartSetBlock()`. Then, that extent is wrapped in change set, masking, and limiting extents, and a final event is posted with a stage of `Stage.BEFORE_HISTORY`. The resulting extent is used for the normal `EditSession.setBlock()`.

Edit Session Event

The `EditSessionEvent` is the way to hook into WorldEdit's changes in an efficient manner. It is fired as a part of `EditSession` creation, and allows hooking into the extent stack at various steps as described above. You can subscribe to this event by registering with WorldEdit's event bus:

```
WorldEdit.getInstance().getEventBus().register(new Object() /* [1] */ {
    // Make sure you import WorldEdit's @Subscribe!
    @Subscribe
    public void onEditSessionEvent(EditSessionEvent event) {
        if (event.getStage() == /* the stage you're interested in */) {
            event.setExtent(new MyCustomExtent(event.getExtent()));
        }
    }
});

// [1]: You don't need to use an anonymous class, you can also store your method in its␣
↪own separate class
//      and construct it here instead.
```

When creating your own extent class, consider extending `AbstractDelegateExtent`, which allows you to only override the methods you're interested in. Don't forget to call the appropriate `super` method to actually perform the changes!

Adapters

WorldEdit works across many Minecraft modding platforms. This implies that WorldEdit's API does not use any platform's API types, such as Bukkit's `Player` or Sponge's `World`. Instead, WorldEdit has its own set of API types, and the platform-specific library (see *API Libraries*) contains an *adapter* class to turn the platform's API types into WorldEdit's API types, and vice versa. For example, you can turn an `org.bukkit.entity.Player` into a `com.sk89q.worldedit.entity.Player` like so

```
org.bukkit.entity.Player player = /* get a player */;
Player wePlayer = BukkitAdapter.adapt(player);
```

Nearly every other WorldEdit type (such as `World`, `BlockVector3`, or `BlockState`) has a similar conversion to and from the platform type. These are best discovered by looking at the methods in the adapter classes in your IDE.

1.7.2 API Examples

These examples represent common use-cases of the WorldEdit API.

Clipboard Examples

> **Note**
>
> This documentation covers the API for using clipboards. See *Clipboard* for in-game usage & explanations of what clipboards are.

Concepts used in these examples: *Regions*, *Edit Sessions*, *Extents*

Copying

Copying is the most common way to create a clipboard. To do it, you'll need a `Region`, and a source and target extent, such as a `World` and a `Clipboard`. In this example we use a `CuboidRegion` and the standard `BlockArrayClipboard`. Then, all you need to do is pass the parameters to the `ForwardExtentCopy`, apply configuration (such as calling `setCopyingEntities(true)`) to copy entities), and call `Operations.complete`.

```
CuboidRegion region = new CuboidRegion(min, max);
BlockArrayClipboard clipboard = new BlockArrayClipboard(region);

ForwardExtentCopy forwardExtentCopy = new ForwardExtentCopy(
    world, region, clipboard, region.getMinimumPoint()
);
// configure here
Operations.complete(forwardExtentCopy);
```

You may want to *save* the clipboard after this. Note that if you are only copying a clipboard to paste it immediately again, you should skip making the clipboard entirely. Instead, build an `EditSession` for the target world and pass that to `ForwardExtentCopy` - it is capable of copying blocks between any two extents, or even between the same one, and is not limited to clipboards.

Pasting

Pasting is the only way to move blocks from a `Clipboard` to another `Extent`, typically a `World`. To paste, you'll need a target `Extent` (generally an `EditSession` for a `World`) and a `Clipboard`. Create a `ClipboardHolder` with your clipboard, then get a `PasteBuilder` by calling `createPaste` with the `EditSession`. Call `.to` to set the position at which you want to paste (this will be offset by the clipboard offset, see the clipboard page above for more information).

Add any other configuration you want (masks, paste entities, paste biomes, etc.), and then call `build()` to get an operation. Complete the operation, and all the blocks will be pasted. Note that if you want to rotate the clipboard, you'll need to `setTransform` on the `ClipboardHolder` *before* calling `createPaste`.

Full example:

```
try (EditSession editSession = WorldEdit.getInstance().newEditSession(world)) {
    Operation operation = new ClipboardHolder(clipboard)
            .createPaste(editSession)
            .to(BlockVector3.at(x, y, z))
            // configure here
            .build();
    Operations.complete(operation);
}
```

You may want to *load* a clipboard before this.

Schematic Examples

This section deals with schematics, which are a related but distinct concept. Schematics specifically refer to a saved clipboard, not a clipboard in-memory.

Saving

A `Clipboard` can be saved to disk very easily. All you need is a `ClipboardFormat`, a `Clipboard`, and an `OutputStream`. Then you can call `getWriter` on the format and `write` on the writer with your `Clipboard`. Here's an example for saving a clipboard to file.

```
File file = /* figure out where to save the clipboard */;

try (ClipboardWriter writer = BuiltInClipboardFormat.SPONGE_SCHEMATIC.getWriter(new␣
↪FileOutputStream(file))) {
    writer.write(clipboard);
}
```

Loading

Loading a `Clipboard` is nearly as simple. You can either force a specific `ClipboardFormat`, or have WorldEdit discover the format of the schematic you want to load. The example does the latter. Then you can call `getReader` on the format and `read` on the reader to get a `Clipboard` instance.

```
Clipboard clipboard;

ClipboardFormat format = ClipboardFormats.findByFile(file);
try (ClipboardReader reader = format.getReader(new FileInputStream(file))) {
    clipboard = reader.read();
}
/* use the clipboard here */
```

LocalSession Examples

> **Note**
>
> This documentation covers the API for programmatically accessing session data. See *Sessions* for the in-game explanations of what sessions are. See *Local Sessions* for the general overview of LocalSessions in API.

Concepts used in these examples: *Actors*, *Regions*, *Local Sessions*, *Adapters*.

Getting a LocalSession

Before getting a LocalSession, you'll need to have the *actor* whose session you want. Generally, actors will be obtained by adapting a platform-specific type via the *Adapters*. In the following example, we'll use a Bukkit `org.bukkit.entity.Player` object.

```
org.bukkit.entity.Player player = ...; // platform-specific player class, generally␣
↪obtained from a command, event, etc.
Player actor = BukkitAdapter.adapt(player); // WorldEdit's native Player class extends␣
↪Actor
SessionManager manager = WorldEdit.getInstance().getSessionManager();
LocalSession localSession = manager.get(actor);
```

Now that you have a session, there's various things you may want to do with it.

Getting a player's selection

```
// get a LocalSession as per the above example
LocalSession localSession = ...;
Region region; // declare the region variable
// note: not necessarily the player's current world, see the concepts page
World selectionWorld = localSession.getSelectionWorld();
try {
    if (selectionWorld == null) throw new IncompleteRegionException();
    region = localSession.getSelection(selectionWorld);
} catch (IncompleteRegionException ex) {
    actor.printError(TextComponent.of("Please make a region selection first."));
    return;
}
/* you can now use the region object for edits, check for a specific shape, etc. */
```

Accessing a player's clipboard

As seen in the *Clipboard Examples*, you can copy a region or load a schematic to get a clipboard, and you can paste a clipboard or save it to a schematic. For completely programmatic access, this may be enough, but sometimes you want to interact with a player's clipboard directly. In those cases, you can get or set the player's clipboard through their `LocalSession`.

Example 1: Setting the player's clipboard

```
LocalSession localSession = ...; // get a LocalSession as per the first example
Clipboard clipboard = ...; // load a schematic or copy a region as in the clipboard␣
↪examples
localSession.setClipboard(new ClipboardHolder(clipboard));
```

Example 2: Getting the player's clipboard

```
LocalSession localSession = ...; // get a LocalSession as per the first example
ClipboardHolder clipboard; // declare variable
try {
    clipboard = localSession.getClipboard();
} catch (EmptyClipboardException ex) {
    actor.printError(TextComponent.of("Your clipboard is empty."))
    return;
}
/* you can now paste the clipboard somewhere, save it to a schematic, etc. */

// bonus example: applying rotation to the player's clipboard
AffineTransform transform = new AffineTransform();
clipboard.setTransform(clipboard.getTransform().combine(transform.rotateY(90)));
```

Storing an EditSession in a Player's History

After programmatically creating and using an *EditSession* to change some blocks, you may want to store that edit in the player's history so that they can later use `//undo`.

```
LocalSession localSession = ...; // get a LocalSession as per the first example
EditSession editSession = ...; // previously used edit
localSession.remember(editSession);
```

1.7.3 Internal APIs

Some of WorldEdit is not considered public API and may be changed at any moment without warning. Usage of this code is not considered proper, and will receive no support.

The precise definition of internal API is anything not accessible according to standard Java access rules, and any of the following types:

- Anything in the platform implementations. An exception is the `*Adapter` class for each platform.
- **Anything in the following packages:**
  - `com.sk89q.worldedit.command`
  - `com.sk89q.worldedit.internal`
- Anything explicitly marked as internal.

WorldEdit provides a stable public interface for other mods and plugins to build off of. It provides platform-independent interfaces and classes for working with Minecraft blocks, biomes, and worlds. Limited entity support is present as well.

1.7.4 API Libraries

You can get the API via a Maven repository, compatible with Maven, Gradle, sbt, and many other build systems. The repository is https://maven.enginehub.org/repo/, and WorldEdit is under the group `com.sk89q.worldedit`. Depending on which parts of the API you need, choose one of the following names:

- `worldedit-core`: The core APIs. Does not depend on any platform, and provides no conversion classes.
- `worldedit-bukkit`: The Bukkit implementation. Depends on Bukkit API and provides conversion between Bukkit types and WorldEdit types using `BukkitAdapter`.
- `worldedit-fabric-mcXYZ`: The Fabric implementation. Depends on Fabric API and provides conversion between Fabric types and WorldEdit types using `FabricAdapter`. Replace XYZ with the appropriate Minecraft version.

- `worldedit-neoforge-mcXYZ`: The NeoForge implementation. Depends on NeoForge and provides conversion between NeoForge types and WorldEdit types using `ForgeAdapter`. Replace XYZ with the appropriate Minecraft version.
- `worldedit-sponge-mcXYZ`: The Sponge implementation. Depends on Sponge API and provides conversion between Sponge types and WorldEdit types using `SpongeAdapter`. Replace XYZ with the appropriate Minecraft version. (Only supports up to 1.12.2 due to Sponge only being available up to that version.)
- `worldedit-cli`: The command-line implementation. Probably not very useful as a dependency, but could be used to automatically run WorldEdit outside of Minecraft.

The version is dependent on the version of Minecraft you are building for. 6 is for Minecraft versions 1.12.2 and below, and 7 is for 1.13 and above. Support is only offered for the latest WorldEdit version, and the latest Minecraft version, but generally the API is similar enough across versions that examples are very similar. These documents also only cover the latest version of WorldEdit, although old versions may be reached using the navigator in the bottom right. The exact version matches the version on the downloads page, or you can browse the repository itself: https://maven.enginehub.org/repo/com/sk89q/worldedit/worldedit-core/

To get started with the API, read *API Concepts*. Some common API usages are documented in *API Examples*. When developing, take note of *Internal APIs* to ensure you're using supported APIs. If you need the Javadocs, they are hosted at https://docs.enginehub.org/javadoc/com.sk89q.worldedit/worldedit-core/7.4.0/.

1.8 Common Questions

1.8.1 General

Why don't any commands work?

If no commands work, it may because WorldEdit failed to start:

- Make sure that you are running Bukkit/Forge/Sponge/etc. A vanilla minecraft server will not load plugins/mods!
  - You can run a command such as `version` (Bukkit), `sponge version` (Sponge) or `forge help` (Forge) to ensure your server is running proper software. In single-player, the Main Menu should have a "Mods" button (and WorldEdit should be in the list!).
- Make sure that you have the proper version of WorldEdit for your version of Minecraft.

If those solutions do not help you, you will need to look through your startup log:

- If you use a game server host, use its log viewer.
- You can also open up "latest.log" in the logs folder of your server directory. (On older versions of Minecraft, the log file was "server.log" in the root directory.)

If you are unable to discover the problem from reading the server log, you can *ask for help or submit a bug report*.

How old is WorldEdit?

WorldEdit began in September 2010 for the "hMod" modding platform by sk89q. Later on, WorldEdit was ported to Bukkit, and eventually to Forge and other platforms.

Who works on WorldEdit?

WorldEdit has been developed by many people, and large portions of WorldEdit include contributed code. The list of top contributors can be found on GitHub.

1.8.2 World-Editing

Commands all return "0 blocks changed" even though they should be changing.

If you've previously set a global mask with `//gmask <mask>`, you'll have to clear it again with `//gmask` so it no longer masks your edits.

My server crashes when I do large edits (1 million blocks or more)!

> **Note**
>
> If your client is disconnected / timed out, these instructions will not fix that. At this time, WorldEdit does not attempt to keep clients connected while processing edits. These instructions only allow for the server to actually complete the edit.

The two most likely causes for this are either the watchdog crashing your server, or it running out of memory.

Ensure that you allocate approximately 1 gigabyte (for the general server) + 2 gigabytes for every ten million blocks you are editing. For example, if you're editing around 50 million blocks, divide `50 / 10` to get `5`, then add `1 + 2 * 5` to get 11 gigabytes. To allocate this amount to your server, specify `-Xmx11G`. See these example instructions for Bukkit/Spigot/Paper. They also apply in a similar way to NeoForge/Fabric.

To fix the watchdog crashing the server, try increasing the watchdog timeout for your server. For Spigot/Paper, this is `timeout-time` in the `spigot.yml` file. For NeoForge/Fabric, it is `max-tick-time` in the `server.properties` file. This should not be needed in most cases, however, as WorldEdit already ticks the watchdog automatically.

My client crashes when I do large edits (1 million blocks or more) in single-player!

You can use the same instructions as the server case above, but you don't need to worry about the watchdog. Single-player does not use a watchdog.

How do I remove a tool/brush from the item I'm holding?

Use the `/tool none` or `/brush none` command while holding the item. These are both currently the same command, so it doesn't matter which one you pick.

Why isn't sign text/chest contents/entities/etc working?

On all versions of WorldEdit, you need to have the `worldedit.setnbt` permission (which can be granted through OP) in order to set block entities.

On Bukkit servers, WorldEdit has to use some version-specific adapters to get full access to many functions due to how Bukkit works. It uses these adapters for block entities (blocks that use additional data including signs, containers, etc.), entities, and some other functionality. What this generally means is that every new release of Minecraft will require you to update WorldEdit. Usually, WorldEdit will be updated quickly, and you can find new releases or experimental builds via the links on the *main page*.

When I select, WorldEdit also selects (-1, -1, -1)!

This is due to a conflict with the Enchantable mod by MrCrayFish, described here.

You can either remove the mod, or use the *long range wand* tool.

1.9 Getting Help

If you have a question or have errors,

- Join our Discord (preferred)

If you have a feature request or bug report,

- Submit it to our issue tracker

1.10 Localisation

WorldEdit has inbuilt support for localisation, sourced from community-contributed translations on Crowdin. These translations are included within WorldEdit releases, so all languages are supported out of the box. WorldEdit uses the language set on each player's Minecraft client to handle translations, so every player will see text in the language that they've defined.

This means that on multilingual servers, every player can use their preferred language, rather than having to use a single server-defined language.

1.10.1 How to Contribute

If some parts of WorldEdit aren't fully translated in your language, this usually means that no one has yet submitted a translation.

We use Crowdin to facilitate translation submissions, with our Crowdin page available here. Each time we cut a new WorldEdit release, we pull in the new translations from the website.

Contributions to our Crowdin are strongly appreciated, as everyone benefits from better translation coverage.

> **Note**
>
> At the time of writing, some lines of text relating to command descriptions cannot be translated due to limitations within the command library. Once this is resolved, we'll be sure to add that text into our translation system.

1.10.2 Custom translations

WorldEdit supports loading user-provided translations, from the `lang` folder in the WorldEdit directory.

When loading languages, a priority system is used to ensure that all available localisation information is loaded. This is done on a per-string basis, so it's possible for some translations to load from the WorldEdit directory, and others to load from the jar file. Therefore, if a user provided translation only provided a translation for one single string, it'd still load all of the missing entries from the internal translation data. That way your local copy only needs to contain translations for strings you are explicitly overriding, rather than having to keep it up to date with every WorldEdit release.

The search order for strings works as follows:

1. `lang/[language-code]/strings.json` within the WorldEdit folder within the game directory.
2. `[language-code]/strings.json` in `lang/i18n.zip` within the WorldEdit folder within the game directory.
3. `[language-code]/strings.json` in `lang/i18n.zip` within the WorldEdit jar file.
4. `lang/strings.json` within the WorldEdit jar file (only for the default language).

If for whatever reason a string in that language cannot be found, it'll repeat the process with a less specific language (eg, `fr` instead of `fr-CA`), and finally fall back to the default language (`en`).

Due to the way this priority system works, it's recommended to *only* include strings that you are actually intending to modify within your user directories, as that will allow you to automatically benefit from future improvements in later WorldEdit updates.

> **Note**
>
> This system is mostly provided to allow you to fill in the gaps of common languages on your Minecraft server, not to fully customise messages. While you can customise messages this way, there are some limits to what can be changed such as an inability to change colours.

As a reference for the structure of the files, you can find the `lang/i18n.zip` file within the WorldEdit jar file. Any custom translations will follow this same layout, either within a zip file, or in the `lang` folder.

Zip method

Placing a zip file at `WorldEdit/lang/i18n.zip` within your game directory will cause WorldEdit to load from this file. Strings from this zip file will take priority over those within the WorldEdit jar file.

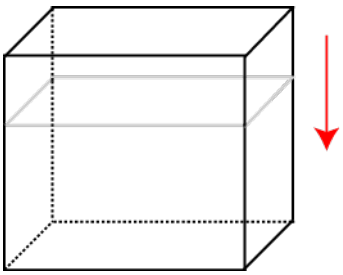
File method

Rather than using a zip file, you can instead directly store the locale files within folders. This would mean that Australian English would be in the `lang/en-AU` folder, or Greek would be in the `lang/el` folder. The "base" translations (`en`) would live in a `lang/strings.json` file, rather than having their own directory. This is because they don't come from Crowdin, and instead are written by us when writing the plugin.

1.11 Source Code

You can find the source code to WorldEdit on GitHub.

WorldEdit is open source. Contributions must be licensed under the GNU General Public License v3.



CHAPTER

TWO

LINKS

- WorldEdit Homepage
- Downloads for Bukkit/NeoForge/Fabric
- Downloads for Sponge
- Experimental Builds
- Discord Server