Github, Packwiz and Unsup tutorial for Windows users

Github (basics)

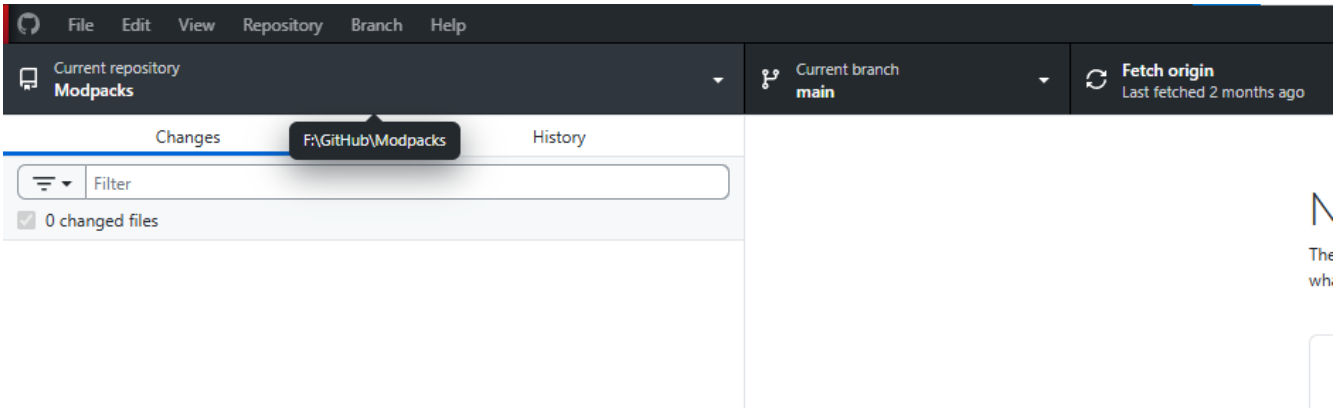
(Notepad++ or another text editor better than basic Notepad is *highly* advised for this)

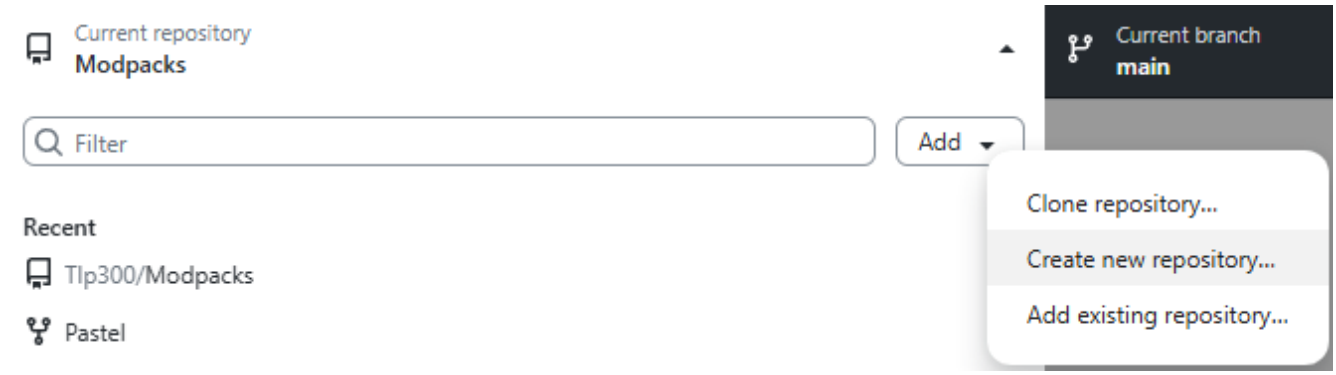
The full usage of Git and Github is *far* too complex for a single document, so this will be focused around the very basics, using the official Github Desktop GUI. Committing and pushing.

First off, make a Github account and set up two factor authentication. Having packs on Github counts as “contributing code”, so 2FA is required.

Next download and install Github Desktop, and sign in through the File>Options>Accounts button in the top left.

Click the Repository Managment button just below that, and create a new repository.





Name it whatever you wish

After that, there will be a new folder in the path you set. Open it, and create a new folder by the name of your modpack, and *another* folder in that called pack. Your Packwiz modpack will go in that folder.

Go back to the folder Github created, and open the .gitattributes file. If it doesn't exist, create it. Put into it

* -text
* -crlf

This should disable Git's automatic line ending conversion, which only occurs on Windows, and breaks Packwiz on upload by changing the hash of every file.

```
# Disable Git line ending conversion, to prevent packwiz index hashes changing when committing from Windo
* -text
* -crlf
```

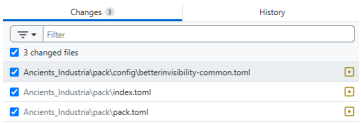
Then go to your Windows user folder – should be C:\Users\<username> and open the .gitconfig file. There, put

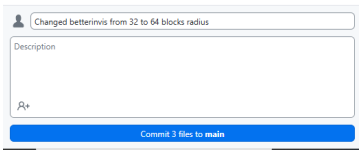
[core]
    autocrlf = false

This does the exact same thing, but globally. Redundancy is good.

```
[core]
    autocrlf = false
[filter "lfs"]
    clean = git-lfs clean -- %f
    smudge = git-lfs smudge -- %f
    process = git-lfs filter-process
    required = true
```

After all that, the repo is fully set up and ready to use. Any change to any file in the folder should show a commit preview, and to commit simply type in a name (typically something like "Added X mod", or "Changed Y config") and press the Commit button





Then just press the Push Origin button to push the changes, uploading them to Github.

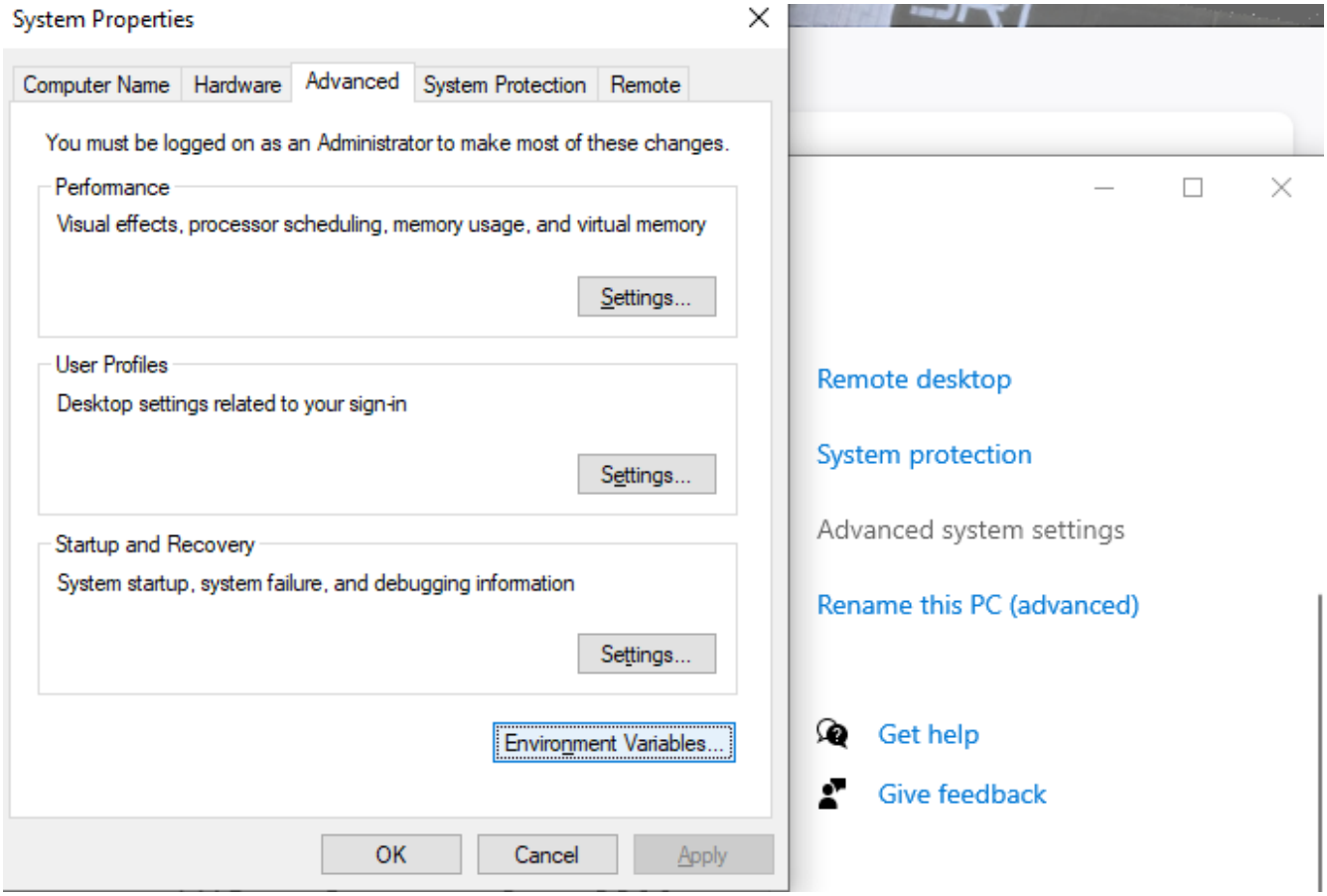
Commits should be done often, after every change, similar to saving a document. Pushing only need be done when you're ready to upload your changes.

Packwiz

Setting up Packwiz

Download Packwiz from Github Actions, or failing that, from my Github repo that contains the latest packwiz executable for Windows at the time of writing (the .exe)

Place the .exe file somewhere – it doesn't matter where. For this example, it will be going in E:\Programs\packwiz. Open the System Properties GUI (Control Panel>System and Security>System>Advanced System Settings on the right bar, OR search "Edit the system environment variables" in Windows search), and click "Environment Variables"



Double click the entry named “Path”, then click “New”. Put in the path of the packwiz.exe, which in this example would be E:\Programs\packwiz, then click “OK” on all the windows until they’re all closed. That is done and no more needs to be done with that file or folder again.

Creating a pack

Packwiz is a command line tool, which means it does not have a GUI. To use it, it must be interfaced with through a terminal of some sort – Powershell is adequate for this, and is always available. Open Powershell by right-clicking the start button and clicking “Windows Powershell”, or by just searching for it in Windows search.

Navigate to your Github modpack folder (this will use `F:\GitHub\Modpacks\examplepack\pack`) with the `cd` command, used as `cd F:\GitHub\Modpacks\examplepack\pack`. Any folder will work, however, but make sure it’s empty to start with.

After that, Packwiz itself can be used. Packwiz has several different commands, which will be gone over in more detail at the end of the document. To start, use packwiz init and follow the prompts to create an empty modpack.

Basic Information

Packwiz uses “payload files” instead of mod files. These payload files point to where the mod/resourcepack/datapack/other can be downloaded from, and contain a sha212 hash of the file to ensure it hasn’t been tampered with. They can be identified by their suffix of `.pw.toml`, and wherever the payload file is, is where the file it points to will be downloaded in the actual pack. Payload files also contain a “side”, which allows Packwiz to know if they’re used only on the client, only on servers, or both. Non-payload files can also be used, and are generally things like config files, custom datapacks, scripts and other such pack-specific files. These always default to both for sidedness.

It also has two more files – `pack.toml` and `index.toml`. `index.toml` contains the location of and a hash of every file in the pack, and `pack.toml` contains information about the pack (loader, loader version, MC version, pack name etc) as well as a hash of `index.toml`.

Because of this hashing system, after *any* modification to the pack that isn’t done through the command line, `packwiz refresh` will need to be run to ensure all hashes are up to date. Failure to do so will result in a hash mismatch, meaning Packwiz will not know where to look for the files, and Unsup will refuse to download them out of fear of tampering. This includes updating config files, editing datapacks/scripts, manually changing the sidedness of a mod or moving a file.

The structure of a packwiz pack is *exactly* like that of the finished and installed modpack, so anywhere a config file, datapack, resourcepack or mod payload is put will be where it ends up on the user’s pack instance.

Building your pack

Now that the pack is created, mods can be added. Packwiz has four different commands for this, for adding mods from Curseforge, Modrinth, Github releases and finally a file from *any* URL on the internet. Do note however that if you're intending to release your pack on Curseforge or Modrinth, created exports will *not* include mods from sites other than that one as part of the pack, but just download and include the mod file in the export itself.

The best way to specify a mod from either Curseforge or Modrinth is with its URL – other terms, such as name or ID may be used, but are either too general or slightly more difficult to get than just copying the URL. The examples used here will be Pastel.

`packwiz modrinth add https://modrinth.com/mod/pastel-mod` will add the latest version of the mod from Modrinth, and `packwiz modrinth add https://www.curseforge.com/minecraft/mc-mods/pastel` adds the latest version of the mod from Curseforge. For both, dependencies are automatically checked and, if not already added *from that site*, asked if they should also be added. Packwiz cannot detect if dependencies are already added from another site. To specify a version, use the URL of that version, so `packwiz modrinth add https://modrinth.com/mod/pastel-mod/version/1.1.3` or `packwiz curseforge add https://www.curseforge.com/minecraft/mc-mods/pastel/files/6847961`. Mods added from Modrinth will be automatically set to download on client, server or both depending on what they are labelled as, but mods added from Curseforge will default to both sides.

`packwiz github add` uses the URL of a github repository and takes the latest release

`packwiz url add` requires both the URL of a file (as in, when you enter it in a browser a download starts) and a name. For example, `packwiz url add ProjectBlue https://github.com/Egassy/Special-pack-files/raw/refs/heads/main/1.7.10/jars/ProjectBlue-1.1.4-mc1.7.10.jar` would add the file `ProjectBlue-1.1.4-mc1.7.10.jar` with the payload file name of `ProjectBlue.pw.toml`.

Packwiz commands

All commands are prefixed with `packwiz`, which will show up in yellow in Powershell

- `modrinth`
  - `add <url, slug, or ID>`: Adds a mod, resourcepack or datapack from Modrinth
  - `export`: Exports to a Modrinth .mrpack

- curseforge
  - add <url, slug, or ID>: Adds a mod, resourcepack or datapack from Curseforge
  - export: Exports to a Curseforge .zip. Add --side client to only export client and both sided mods, and --side server to only export server and both sided mods (creating a serverpack)
  - import <file path, including file>: Attempts to automatically convert a Curseforge .zip to a Packwiz pack. Overrides and can be used in place of init.
  - detect: Finds .jar files in the mods folder of the pack and attempts find them on Curseforge, converting them to payload files. Experimental and a WIP.
  - open <payload file name>: Opens the Curseforge page of a payload file in the default web browser.
- github
  - add <repo URL>: Adds a mod from Github releases. Use --branch <branch name> to specify a branch
- url
  - add <name> <URL>: Adds any file from a URL. Use --force to force it even if it would be valid for adding through Curseforge or Modrinth
- refresh: reruns hashing and checks for added or removed files through the entire pack.
- remove <payload file name>: Cleanly removes the mod from the pack
- list: Generates a list of all mods, datapacks and resourcepacks in the pack.
- update: <payload file name>: Updates that mod from Curseforge, Modrinth or the latest Github release. Use --all instead to update all unpinned mods
- pin <payload file name>: Pin a file, excluding it from update --all
- unpin <payload file name>: Unpins a file
- migrate
  - loader <version, latest, recommended>: Updates the loader to a specific version, to the latest available for that loader on that MC version, or to the recommended loader version for that MC version.
  - minecraft <version>: Updates Minecraft to a specific version. Use with care
- serve: creates a locally-hosted “development server” that has the packwiz files on it, and automatically refreshes each time it’s accessed. Defaults to port 8080, use --port <number> to change that. Very useful for testing with Unsup. Default pack.toml location is http://localhost:8080/pack.toml

Unsup

Basic Information

Unsup is an automatic Java-based updater/directory syncer that can piggyback off of Minecraft's Java launch and has specific compatibility with Packwiz. This means that a modpack using Unsup can be pointed at a Packwiz pack in a Github repository, and any changes in the repo will be applied to the users pack at launch.

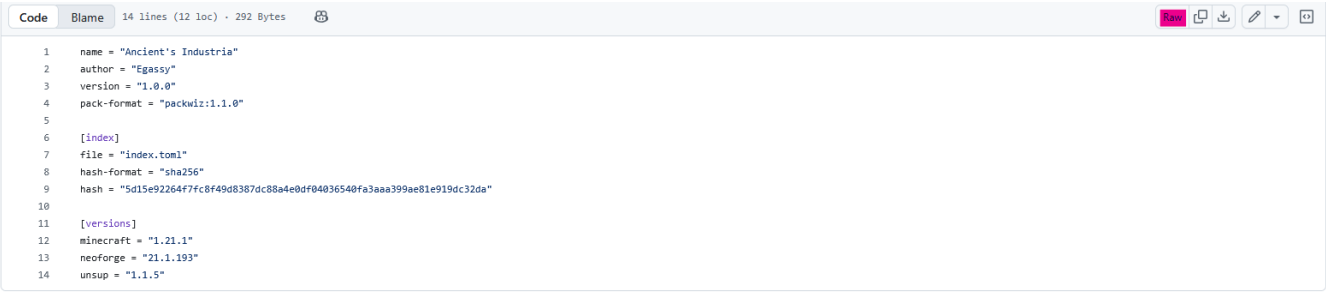
Basic Usage

Download an Unsup jar from https://modrinth.com/mod/unsup or https://git.sleeping.town/unascribed/unsup/releases. Place it in the *root* of the instance folder, or in the instance's .minecraft folder if you are using a MultiMC based launcher such as Prism. Alongside it, put a file named unsup.ini containing *at least* the following:

```
version=1
preset=minecraft

source_format=packwiz
source=<source URL>
[colors]
progress=b017cf
button=4e15b0
```

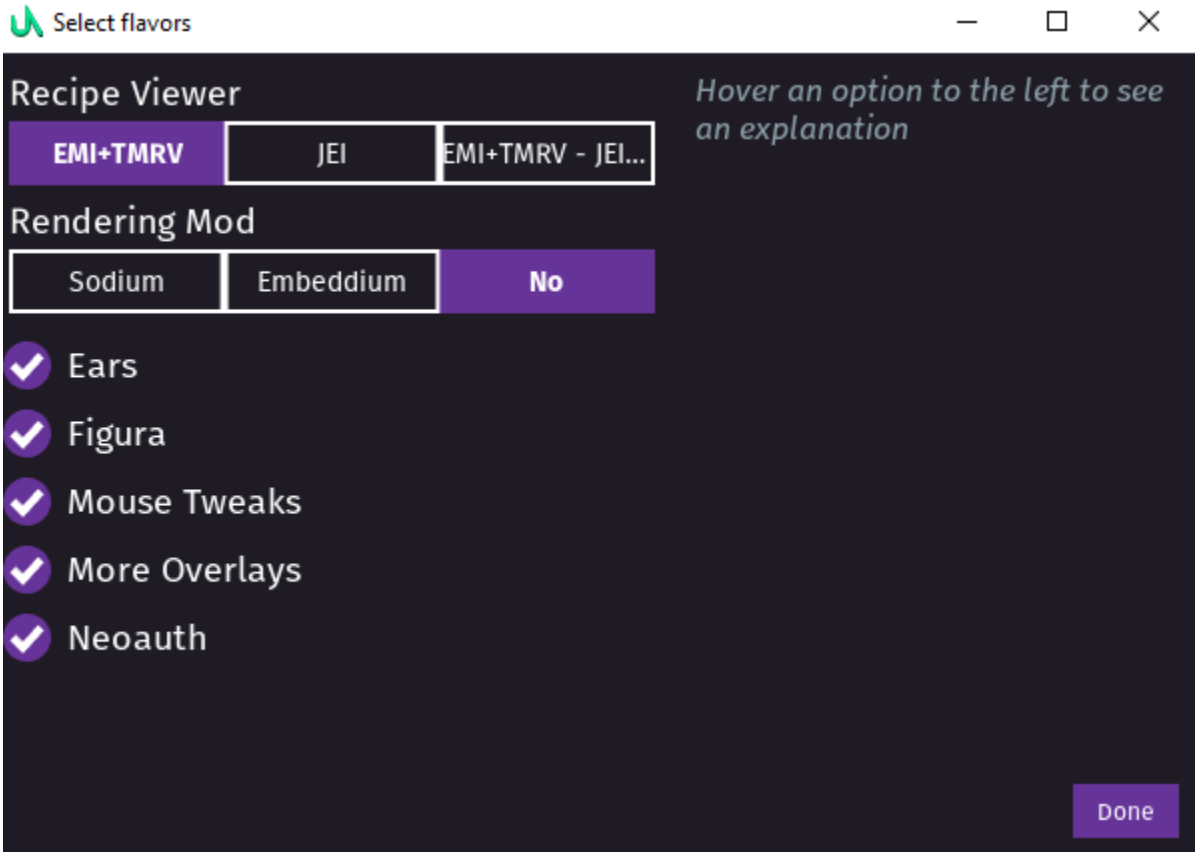
The <source URL> needs to be the *raw* URL of an uploaded pack.toml. Using something that displays a formatted or filtered file *will not work*. To get this from Github, open the pack.toml, right click the "Raw" button and copy the given link.



Then just add -javaagent:<name of unsup jar> to your javaargs for that instance. At time of writing, that would be -javaagent:unsup-1.2.0-pre1.jar. This works on servers too, if no_gui=true is added after preset=minecraft.

Flavours

Unsup supports “flavours”, sets of one or more optional downloads (not necessarily mods!) that can be selected in a GUI that appears before downloading anything. Flavours allow optional mods, resourcepacks, datapacks, shaders and... anything, really, to be part of a pack, including having multiple-choice options (for example, using Xaero’s Minimap *and* World Map, *or* Journeymap, *or* Voxelmap *and* Liteloader)

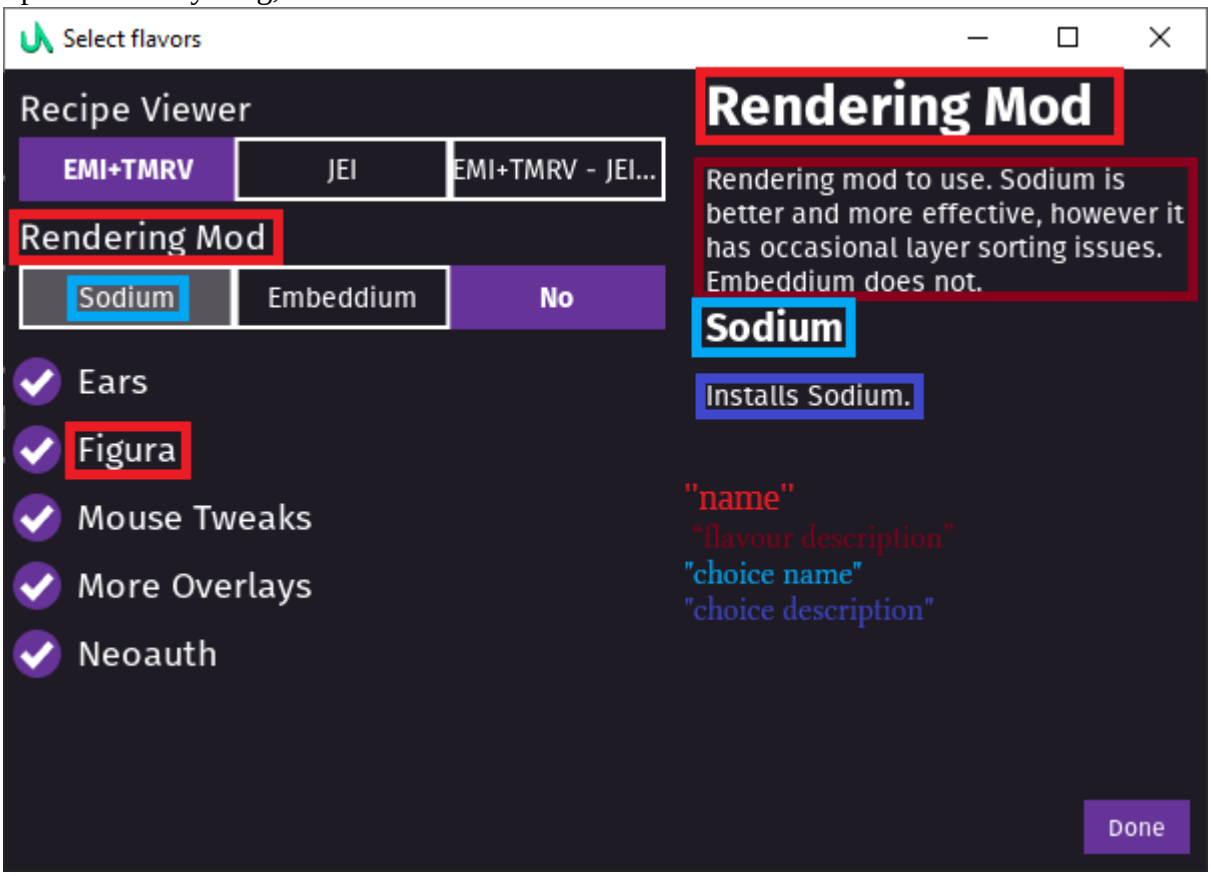


Flavours are set in an unsup.toml file in the same folder as pack.toml. They follow the format

```
[flavor_groups.groupid]
name = “name”
description = “flavour description”
side = “side”
[[flavor_groups.groupid.choices]]
id = “choiceid”
name = “choice name”
description = “choice  description”
[[flavor_groups.groupid.choices]]
id = “choiceid2”
name = “choice name2”
description = “choice description2”

[metafile.metafilename]
flavors = "choiceid"
```

groupid can be anything, it doesn’t matter what



choices is required

“side” declares if that flavour applies to client, server or both (options are “client”, “server” or “both”)

“name” is the displayed name of the flavour in the GUI when hovering over any choice in that flavour

“flavour description” is what appears below the flavour name and should describe the overall flavour

“choice name” appears below the flavour description when hovering over that specific choice

“choice description” appears below the choice name when hovering over that specific choice and should be used to describe what that specific choice does

“choiceid” is the internal ID of that choice, and can be anything

metafilename is the name of the packwiz .pw.toml file

Having a flavour with two choices, with the ids being “<groupid>_on” and “<groupid>_off” and the names being “Yes” and “No” will result in the flavour appearing as a checkbox

In order to make unsup recognize the existence of a flavour file, a single line must be added to the end of pack.toml (in the [versions] section), reading unsup = "unsup version", with “unsup version” being the version of unsup meant to be in use.

For reference:

```
[flavor_groups.viewer]
name = "Recipe Viewer"
description = "Recipe viewer mod to use. EMI (with the TMRV compat layer) is more performant, more customizable and has more features, and allows faster load times by loading data on world load. JEI has more commonly used direct compatability instead of requiring a universal compat layer mod."
side = "both"
[[flavor_groups.viewer.choices]]
id = "viewer_emi"
name = "EMI+TMRV"
description = "Installs EMI and TMRV."
[[flavor_groups.viewer.choices]]
id = "viewer_jei"
name = "JEI"
description = "Installs JEI."
[[flavor_groups.viewer.choices]]
id = "viewer_emi_configed"
name = "EMI+TMRV - JEI edition"
description = "Installs EMI and TMRV with customized configs to make it look like JEI."

[flavor_groups.rendering]
name = "Rendering Mod"
description = "Rendering mod to use. Sodium is better and more effective, however it has occasional layer sorting issues. Embeddium does not."
side = "client"
[[flavor_groups.rendering.choices]]
id = "rendering_sodium"
name = "Sodium"
description = "Installs Sodium."
[[flavor_groups.rendering.choices]]
id = "rendering_embeddium"
name = "Embeddium"
description = "Installs Embeddium."
[[flavor_groups.rendering.choices]]
id = "rendering_none"
name = "No"
```

```
description = "Do not install a rendering mod."

[flavor_groups.ears]
name = "Ears"
description = "Uses the otherwise-blank space on a modern 64x64 skin in combination
with alpha layer encoding to give extra features to your player model, such as
ears, wings, a custom cape, breasts, a snout and more. Client only. No effect
(other than seeing other players Ears skins) if you do not have an Ears skin.
Client only."
side = "client"
[[flavor_groups.ears.choices]]
id = "ears_on"
name = "Yes"
description = "Installs Ears."
[[flavor_groups.ears.choices]]
id = "ears_off"
name = "No"
description = "Do not install Ears."

[flavor_groups.figura]
name = "Figura"
description = "Animation-capable player models that come directly from Blockbench,
with near unlimited capabilities for customization. Client only."
side = "client"
[[flavor_groups.figura.choices]]
id = "figura_on"
name = "Yes"
description = "Installs Figura."
[[flavor_groups.figura.choices]]
id = "figura_off"
name = "No"
description = "Do not install Figura."

[flavor_groups.mousetweaks]
name = "Mouse Tweaks"
description = "It's Mouse Tweaks. Either you know what it does, or you're living
under a rock."
side = "both"
[[flavor_groups.mousetweaks.choices]]
id = "mousetweaks_on"
name = "Yes"
description = "Installs Mouse Tweaks."
[[flavor_groups.mousetweaks.choices]]
id = "mousetweaks_off"
name = "No"
description = "Do not install Mouse Tweaks."

[flavor_groups.mud]
name = "More Overlays"
description = "Classical light and chunk overlays toggled with F7 and F9, just like
NEIs used to be. Client only."
side = "client"
[[flavor_groups.mud.choices]]
id = "mud_on"
name = "Yes"
description = "Installs More Overlays."
[[flavor_groups.mud.choices]]
id = "mud_off"
name = "No"
```

```
description = "Do not install More Overlays."

[flavor_groups.neoauth]
name = "Neoauth"
description = "Reauthentication without closing the game. Client only."
side = "client"
[[flavor_groups.neoauth.choices]]
id = "neoauth_on"
name = "Yes"
description = "Installs Neoauth."
[[flavor_groups.neoauth.choices]]
id = "neoauth_off"
name = "No"
description = "Do not install Neoauth."


[metafile.emi]
flavors = ["viewer_emi", "viewer_emi_configed"]
[metafile.tmrv]
flavors = ["viewer_emi", "viewer_emi_configed"]
[metafile.jei]
flavors = "viewer_jei"
[metafile.jeimode]
flavors = "viewer_emi_configed"
[metafile.standardmode]
flavors = "viewer_emi"
[metafile.sodium]
flavors = "rendering_sodium"
[metafile.embeddium]
flavors = "rendering_embeddium"
[metafile.ears]
flavors = "ears_on"
[metafile.figura]
flavors = "figura_on"
[metafile.mouse-tweaks]
flavors = "mousetweaks_on"
[metafile.more-overlays-updated]
flavors = "mud_on"
[metafile.neoauth]
flavors = "neoauth_on"
```

Creates the flavour settings seen above

To define preset flavour options, add a section to unsup.ini called [flavors] and set the various flavors to groupid=choiceid

Example:

```
[flavors]
ears=ears_on
mud=mud_on
neoauth=neoauth_off
```

The various configurations are far too extensive to list here, and can be found at:

https://git.sleeping.town/unascribed/unsup/wiki/Config-format for the unsup.ini configuration options (https://git.sleeping.town/unascribed/unsup/wiki/Config-format#colors for puppet GUI colours)

https://git.sleeping.town/unascribed/unsup/wiki/Packwiz-extensions for extended flavour information

https://git.sleeping.town/unascribed/unsup/wiki/Manifest-format if you wish to create your own Unsup pack without relying on packwiz