**Drupal Installation and Basic Setup Guide:**

##### **You said:**

firstly I want a project that is drupal version 9. secondly I don't want to configure apache server. I just want to run a cmd command like php -S localhost:8080 -t \web to run my project. Give me that approach.

**Response:**

You want:

* Drupal 9 project
* No Apache configuration, no virtualhost — just use PHP’s built-in server (using php -S) to run locally.

Perfect for practicing and development.  
 Let’s do it your way — I’ll guide you step-by-step.

# **🛠 Step-by-Step: Create a Drupal 9 Project (without Apache)**

## **1. Install Needed Tools (if not installed)**

Install **Composer** and **PHP extensions** needed for Drupal:

1. sudo apt update
2. sudo apt install php php-mysql php-gd php-xml php-mbstring php-curl php-zip php-xmlrpc php-json php-cli php-common unzip composer

Done? Good.  
(If composer already installed, skip.)

## **2. Create a Drupal 9 project using Composer**

Now create your Drupal 9 project:

1. composer create-project drupal/recommended-project:^9 drupal9practice

* This forces Drupal 9 (because :^9 tells Composer to get the latest 9.x version)
* It will create a folder drupal9practice

Now you have your project ready.

3. Go inside your project:

1. cd drupal9practice

You are now inside the project.

## **4. Start PHP built-in server**

You need to serve the **web** directory, because Drupal puts the entry point inside web/ folder.

Run this:

1. php -S localhost:8080 -t web

**Breakdown:**

* php -S = start PHP’s internal web server
* localhost:8080 = available at [http://localhost:8080](http://localhost:8080/)
* -t web = serve the web folder (Drupal's public directory)

Now you have your Drupal 9 site running at:

[http://localhost:8080](http://localhost:8080/)

## **5. Install Drupal via Browser**

When you visit [http://localhost:8080](http://localhost:8080/), the Drupal **installer screen** will appear.

At database setup step:

* You need to have a database ready.
* You can create it like:

1. sudo mysql -u root –p

Inside MySQL:

1. CREATE DATABASE drupal9db;

(You can just use root user for database during install — since it's for practice.)

Then complete the installation as usual.

⚡ Full Quick Commands Summary

1. sudo apt update
2. sudo apt install php php-mysql php-gd php-xml php-mbstring php-curl php-zip php-xmlrpc php-json php-cli php-common unzip composer
3. composer create-project drupal/recommended-project:^9 drupal9practice
4. cd drupal9practice
5. php -S localhost:8080 -t web

# **Important Points**

* Drupal installer will complain if PHP extensions are missing — install them.
* Always run PHP built-in server **from project root** with -t web.
* You don’t need Apache, VirtualHost, SSL, nothing — just your terminal.
* For production servers, you **must** use Apache or Nginx. (but for practice, PHP server is fine.)

# **Recap:**

|  |  |
| --- | --- |
| **What You Want** | **How We Set It Up** |
| Drupal 9 only | composer create-project drupal/recommended-project:^9 |
| No Apache | php -S localhost:8080 -t web |
| Simple practice server | Built-in PHP server |

##### **You said:**

Got an intermediate task, make me a git ignore file for this project so that I can make a git repo and store this project in github.

**Response:**

Perfect!  
 Before you upload the Drupal 9 project to GitHub, you definitely should have a good .gitignore file — otherwise vendor files, temporary files, private settings etc. will clog your repo.

Here's the correct .gitignore for a Drupal 9 project based on Composer installation:

**.gitignore for your Drupal 9 project**

# Ignore core Drupal stuff that shouldn't go to Git

/web/sites/default/settings.php

/web/sites/default/settings.local.php

/web/sites/default/services.yml

/web/sites/default/files/

# Ignore Composer vendor directory

/vendor/

# Ignore contributed modules/themes downloaded via Composer

/web/modules/contrib/

/web/themes/contrib/

/web/profiles/contrib/

/web/libraries/

# Ignore custom modules/themes if you want to separately version them

# (Uncomment if you want)

/web/modules/custom/

/web/themes/custom/

# Ignore temporary files

/tmp/

\*.swp

\*.swo

\*.DS\_Store

# Ignore PHPStorm, VSCode, other editor settings

.idea/

.vscode/

\*.sublime-workspace

\*.sublime-project

# Ignore node\_modules if you use frontend tooling (optional)

/node\_modules/

# Ignore npm/yarn lock if you don't want to version it (optional)

package-lock.json

yarn.lock

# Ignore backups

\*.bak

\*.tmp

# Ignore test outputs

/results/

phpunit.xml

phpunit.xml.dist

/.phpunit.result.cache

# Misc

/web/.ht.router.php

/web/sites/simpletest/

/core/assets/vendor/

# Ignore database dumps if any

\*.sql

\*.sqlite

\*.tar.gz

# **Why These?**

|  |  |
| --- | --- |
| **Ignore This** | **Because** |
| vendor/ | Composer will re-generate it (composer install) |
| settings.php | Site-specific config, contains DB passwords |
| files/ | User-uploaded files, usually huge |
| modules/contrib/ | Contributed modules should be managed by Composer |
| .idea/, .vscode/ | Developer's personal IDE settings |
| tmp/, \*.swp | Temp files |

**You said:**

I want to store any custom modules and any installed modules and themes

**Response:**

Perfect, then let's adjust your .gitignore slightly based on your goal:

✅ You want to store:

* Your custom modules (web/modules/custom/)
* Contributed modules (web/modules/contrib/)
* Any installed themes (web/themes/contrib/ and web/themes/custom/)

✅ You still want to ignore:

* Temporary files (like /files/, /tmp/)
* Local environment config files (like settings.php)
* vendor/ (because it's regenerated by composer install)

**Updated .gitignore for Your Plan**

# Ignore settings files

/web/sites/default/settings.php

/web/sites/default/settings.local.php

/web/sites/default/services.yml

# Ignore user-uploaded files

/web/sites/default/files/

# Ignore Composer vendor directory

/vendor/

# Ignore temporary files

/tmp/

\*.swp

\*.swo

\*.DS\_Store

# Ignore editor-specific files

.idea/

.vscode/

\*.sublime-workspace

\*.sublime-project

# Ignore frontend tools if you use them (optional)

/node\_modules/

package-lock.json

yarn.lock

# Ignore test outputs

/results/

phpunit.xml

phpunit.xml.dist

/.phpunit.result.cache

# Misc

/web/.ht.router.php

/web/sites/simpletest/

/core/assets/vendor/

# Ignore database dumps

\*.sql

\*.sqlite

\*.tar.gz

# **⚡ Explanation of Changes**

* ❌ No longer ignoring:
* /web/modules/contrib/
* /web/modules/custom/
* /web/themes/contrib/
* /web/themes/custom/
* ✅ So **your installed modules and themes** will be committed into Git.

# **🛠 How your Git structure will look**

|  |  |  |
| --- | --- | --- |
| **Folder** | **Committed to Git?** | **Purpose** |
| web/modules/contrib/ | ✅ | Downloaded modules (e.g., Pathauto, Token, etc.) |
| web/modules/custom/ | ✅ | Your own custom modules |
| web/themes/contrib/ | ✅ | Downloaded themes (e.g., Bootstrap, Barrio) |
| web/themes/custom/ | ✅ | Your own custom themes |
| vendor/ | ❌ | Composer will rebuild it |
| web/sites/default/files/ | ❌ | Uploads, private files |

# **🧠 Important Developer Note**

* **Contrib modules and themes** are usually better managed by **Composer**.
* If you manually download contrib modules (like from drupal.org and copy-paste), then yes you need to version them manually.
* But if you use Composer (composer require drupal/pathauto), you **don't need to commit contrib modules** — Composer.json will take care of it.

👉 **Which style are you using?**

* **Composer style** (install modules via composer) — cleaner, modern way
* **Manual style** (download modules manually) — old-school, still fine for small projects

I can adapt the .gitignore for you based on which style you prefer. 🔥

# **🚀 In short**

✅ Use the updated .gitignore above.

✅ After that:

1. git init
2. git add .
3. git commit -m "Initial commit: Drupal 9 practice site"
4. git remote add origin https://github.com/yourusername/yourrepo.git
5. git push -u origin main

# **🎯 Final Tip for Clean Projects**

You may want to also commit your **composer.json** and **composer.lock** files because:

* composer.json = lists required packages
* composer.lock = locks exact versions

So someone else (or future you) can just clone and do:

1. composer install

**You said:**

ok then remove the contrib related file from gitignore. I will most probably will want to install custom modules with composer. Also tell me this will removing web/sites/default/files break my project when I pull the project and reinstall my project from git?

**Response:**

# **🛠 Updated .gitignore based on your plan**

✅ You want:

* To **install contributed modules and themes via Composer** (correct way)
* To **store your custom modules and custom themes** (your own work)
* To **not** manually commit contrib modules/themes
* To **ignore** web/sites/default/files/

So here's your clean .gitignore now:

# Ignore settings files (they have passwords, etc.)

/web/sites/default/settings.php

/web/sites/default/settings.local.php

/web/sites/default/services.yml

# Ignore user-uploaded files

/web/sites/default/files/

# Ignore Composer vendor directory

/vendor/

# Ignore temporary files

/tmp/

\*.swp

\*.swo

\*.DS\_Store

# Ignore editor-specific files

.idea/

.vscode/

\*.sublime-workspace

\*.sublime-project

# Ignore frontend tools if you use them (optional)

/node\_modules/

package-lock.json

yarn.lock

# Ignore test outputs

/results/

phpunit.xml

phpunit.xml.dist

/.phpunit.result.cache

# Misc

/web/.ht.router.php

/web/sites/simpletest/

/core/assets/vendor/

# Ignore database dumps

\*.sql

\*.sqlite

\*.tar.gz

Now:

* **Custom modules** (web/modules/custom/) ➔ Will be committed
* **Custom themes** (web/themes/custom/) ➔ Will be committed
* **Contrib modules/themes** (web/modules/contrib/, web/themes/contrib/) ➔ Installed by Composer (composer require) — not manually committed
* **Files folder** (/web/sites/default/files/) ➔ Ignored

# **🧠 Now your Important Question:**

Will removing (ignoring) web/sites/default/files break my project when I pull and reinstall?

👉 **Short Answer:  
No, your Drupal will not break — but the uploaded content (like images) will be missing until you recreate or upload them manually.**

👉 **Longer Explanation:**

* /web/sites/default/files/ contains **runtime data** like:
* Uploaded images
* Uploaded PDFs
* Caches
* Private file storage
* **Drupal expects that folder to exist** but if it’s missing:
* You just need to **create the /files folder manually** (empty) or
* Run drush cr (clear cache) and Drupal will recreate it if needed
* **Site installation will complete fine** — no fatal error if /files is missing initially.

✅ Just after pulling from GitHub:

* You manually create web/sites/default/files/ directory if needed:

1. mkdir -p web/sites/default/files

2. chmod 755 web/sites/default/files

3. chown www-data:www-data web/sites/default/files

OR, if you move to real server, you also set correct permissions.

# **📦 How professional teams handle /files/**

|  |  |
| --- | --- |
| **Method** | **How they handle** |
| Small projects | Sometimes manually upload needed files via SFTP |
| Bigger projects | Use separate storage (like S3 bucket or backup /files/ in a separate repo) |
| Team projects | /files/ never in Git — always managed separately |

# **📢 Final Summary for You**

✅ .gitignore is now correct and clean.  
 ✅ Your **code** (modules, themes, settings) is fully in GitHub.  
 ✅ Your **uploads** (/files) are ignored — not a problem; manually recreate if missing.  
 ✅ Your **installed modules/themes** will be managed cleanly by composer.

# **🚀 Next Optional Steps:**

If you want, I can show you next:

* How to **install a module** properly with Composer (composer require drupal/pathauto)
* How to **create your first custom module**
* How to **back up your /files/ separately** easily if you want

**You said:**

I understand now, thanks. Now tell me another thing, If I ignore the settings.php file in the default directory, When I pull the project, where will I get the settings.php file? and also why do we have to ignore this file?

**Response:**

Excellent questions — and honestly, **you're asking exactly the kinds of questions** a good Drupal developer should be asking. 👏  
 Let's go through both questions very clearly:

# **📚 1. If I ignore settings.php, where will I get it when I pull the project?**

👉 When you install Drupal fresh (the very first time), **Drupal copies a default settings file** for you:

* From /web/sites/default/default.settings.php
* To /web/sites/default/settings.php

✅ **The file /web/sites/default/default.settings.php is ALWAYS present** in the Drupal core.  
 ✅ So after pulling the repo:

* You **copy it manually** like this:

1. cp web/sites/default/default.settings.php web/sites/default/settings.php

2. chmod 644 web/sites/default/settings.php

* Then you **edit settings.php** to add database credentials for your local server.

✅ After copying, your site can connect to the database and run normally.

# **🛡️ 2. Why do we ignore settings.php?**

👉 **Main reason: it contains sensitive information!** Inside settings.php, you have things like:

* Database name
* Database username
* Database password
* Trusted host settings (sometimes)
* Local server paths

Example part from settings.php:

$databases['default']['default'] = [

'database' => 'mydb',

'username' => 'mydbuser',

'password' => 'supersecret',

'host' => 'localhost',

'driver' => 'mysql',

];

🔴 **You absolutely do not want to accidentally push your database passwords to GitHub.** Otherwise:

* Hackers can find your database credentials.
* They could attack your database.
* They could steal or delete your site’s data.

👉 So we **never commit** settings.php.  
 Everyone on the team **creates their own local copy** of settings.php, customized to their machine.

# **📜 Professional Flow**

|  |  |
| --- | --- |
| **Step** | **What Happens** |
| Clone the project | Only default.settings.php is available |
| Copy default file | cp web/sites/default/default.settings.php web/sites/default/settings.php |
| Edit | Add your local database settings inside settings.php |
| Local Drupal connects | And site starts running! |

✅ Every developer/team member keeps **their own local settings.php** (not shared).  
 ✅ Common/production settings can be managed separately (e.g., settings.local.php or by Environment Variables).

# **🧠 Bonus Tip**

In big teams, we sometimes **document this in README.md**, like:

After cloning, copy default.settings.php to settings.php, edit your database info, and set file permissions correctly.

That way nobody is confused.

# **🛠 Quick command again for you**

After cloning or pulling:

1. cp web/sites/default/default.settings.php web/sites/default/settings.php
2. chmod 644 web/sites/default/settings.php

# **✅ In Short**

|  |  |
| --- | --- |
| **Question** | **Answer** |
| Where do I get settings.php? | Copy from default.settings.php |
| Why ignore it? | To protect passwords and machine-specific config |

# **🎯 Bonus (if you want)**

Later, if you want **different configs for local/staging/production**, I can show you how Drupal 9 supports **settings.local.php** with automatic loading. (Best practice!)

**You said:**

ok tell me how settings.local.php works.

**Response:**

# **📚 What is settings.local.php?**

* It’s an **optional extra settings file**.
* It is used for **local machine settings** like:
* Local database credentials
* Developer modules (Devel, Kint)
* Disabling CSS/JS aggregation
* Disabling caching
* It **overrides** some parts of your settings.php, but **only locally** — not in production.

✅ It helps you separate **local environment config** from **production config**.

# **🛠 How does Drupal load settings.local.php?**

Drupal core’s default settings.php **already has a built-in line** like this (near the bottom):

php

CopyEdit

if (file\_exists($app\_root . '/' . $site\_path . '/settings.local.php')) {

include $app\_root . '/' . $site\_path . '/settings.local.php';

}

**Meaning:**

* If a file named settings.local.php exists in the same directory as settings.php
* Then Drupal will **automatically load and run it**
* It happens **after** main settings, so it can **override** anything you want.

✅ You don't need to manually load it yourself — it’s automatic if the file exists.

# **🛠 How to set up settings.local.php**

1. **Create a file**:

nano web/sites/default/settings.local.php

2. Add local settings inside it.  
 For example:

<?php

// Local database settings.

$databases['default']['default'] = [

'driver' => 'mysql',

'database' => 'local\_db',

'username' => 'local\_user',

'password' => 'local\_pass',

'host' => '127.0.0.1',

'port' => '',

'prefix' => '',

];

// Disable CSS and JS aggregation for easier debugging.

$config['system.performance']['css']['preprocess'] = FALSE;

$config['system.performance']['js']['preprocess'] = FALSE;

// Disable caching during development.

$settings['cache']['bins']['render'] = 'cache.backend.null';

$settings['cache']['bins']['dynamic\_page\_cache'] = 'cache.backend.null';

$settings['cache']['bins']['page'] = 'cache.backend.null';

// Enable verbose error messages.

$config['system.logging']['error\_level'] = 'verbose';

3. Make sure settings.local.php is ignored by Git

Add in your .gitignore:

/web/sites/default/settings.local.php

✅ So it’s only in your computer, not shared in GitHub.

1. Make sure in settings.php the loading code is uncommented

In settings.php, you should see this near bottom:

if (file\_exists($app\_root . '/' . $site\_path . '/settings.local.php')) {

include $app\_root . '/' . $site\_path . '/settings.local.php';

}

If it's commented (with //), **uncomment it**!

# **📦 Example: How it all works together**

|  |  |
| --- | --- |
| **File** | **Purpose** |
| settings.php | Common settings for everyone (safe) |
| settings.local.php | Personal settings for your machine only (passwords, debug mode, etc.) |

* **On local machine:** both settings.php and settings.local.php run.
* **On production server:** if settings.local.php is missing, **no problem** — production uses only settings.php.

✅ This way you keep GitHub clean and secure, but develop freely on your laptop.

# **🔥 Why is this important?**

* **Local debugging** becomes super easy (disable cache, enable verbose errors).
* **No database credentials leak** into GitHub.
* **Production remains stable** — no dev-only hacks in live server.

# **🚨 Common mistakes to avoid**

|  |  |
| --- | --- |
| **Mistake** | **Why it's bad** |
| Forgetting to add settings.local.php to .gitignore | Accidentally leaks your passwords to GitHub |
| Not uncommenting the include settings.local.php line in settings.php | Drupal won't load your local settings |
| Deleting settings.php thinking settings.local.php is enough | No — settings.php is always required! Local file is just extra |

# **🎯 Quick Setup Recap**

1. Create settings.local.php with your local configs
2. Add /web/sites/default/settings.local.php to .gitignore
3. Uncomment the include line in settings.php
4. Done!

# **✨ Bonus Professional Trick**

If you want to be *super* fancy, you can also do environment detection (local vs staging vs prod) inside settings.local.php using PHP's gethostname() or environment variables.  
 (Not needed now — just telling you what is possible!)

**You said:**

my project setup is done and I have uploaded my project to github. I have another question. If I delete this project from my local machine and pull this project from git right now, what steps do I have to take to reinstall and run the project?

**Response:**

Fantastic! You're moving like a real developer now. 🚀  
 And your question is again **very professional**:

"If I delete my local project and pull from GitHub, what exactly do I need to do to reinstall and run it?"

✅ I’ll answer step-by-step assuming **your current project setup**:

# **📚 Scenario: Fresh Setup from GitHub**

You have a GitHub repository with:

* Your Drupal project files (no vendor/, no web/sites/default/files/, no settings.php)
* A clean .gitignore
* Composer managed contrib modules/themes

Now, imagine you **delete** your local project completely.  
 And you **clone** your GitHub repo fresh.

# **🛠 Step-by-Step to Reinstall and Run**

### **1. Clone the GitHub repo:**

1. git clone https://github.com/yourusername/yourrepo.git
2. cd yourrepo

✅ You have your code, but vendor/ and some other runtime stuff are missing.

### **2. Install project dependencies via Composer**

Because vendor/ is ignored, you need to **rebuild it** using:

1. composer install

✅ This will:

* Download all the necessary PHP libraries
* Download Drupal core
* Download contributed modules/themes as specified in your composer.lock
* Create the vendor/ folder and autoload.php

### **3. Prepare the settings.php file**

Remember, settings.php is **ignored**, so you won't have one.  
 You need to **create it** manually:

1. cp web/sites/default/default.settings.php web/sites/default/settings.php
2. chmod 644 web/sites/default/settings.php

✅ Now you have a basic settings.php file.

### **4. Create files/ directory manually**

Same as above, /files/ folder is ignored.  
 You need to **create it** because Drupal will need it for uploads, caching, etc.

1. mkdir -p web/sites/default/files
2. chmod 755 web/sites/default/files

✅ Now your site can store files properly.

### **5. Set up the database**

You now need a database.

There are two possibilities:

|  |  |
| --- | --- |
| **Situation** | **What to do** |
| You have an existing database backup (.sql) | Import it into MySQL |
| You are starting fresh | Create a new empty database |

Example: Create a new database

1. sudo mysql -u root -p
2. CREATE DATABASE drupal9db;

✅ Now you have a database ready.

### **6. Configure database credentials in settings.php**

Open your settings.php and edit the database settings section:

$databases['default']['default'] = [

'database' => 'drupal9db',

'username' => 'root',

'password' => '',

'host' => '127.0.0.1',

'driver' => 'mysql',

];

✅ Now Drupal knows how to connect to your database.

### **7. Install Drupal (if database is fresh)**

If you have **no site installed yet**, open:

[http://localhost:8080](http://localhost:8080/)

You'll be redirected to the Drupal installer automatically, where you can set:

* Site name
* Admin username/password
* Database info (already set in settings.php, so should skip or auto-connect)

✅ After installation, your site will be ready.

### **8. Start PHP server**

You can now run the site using:

1. php -S localhost:8080 -t web

✅ Site will be running!

# **🧠 Summary Table**

|  |  |
| --- | --- |
| **Step** | **Command** |
| Clone | git clone yourrepo |
| Install dependencies | composer install |
| Create settings.php | cp web/sites/default/default.settings.php web/sites/default/settings.php |
| Create files/ folder | mkdir -p web/sites/default/files |
| Setup DB (optional) | CREATE DATABASE drupal9db; |
| Update DB settings | Edit settings.php |
| Install site (if fresh) | Through browser |
| Start server | php -S localhost:8080 -t web |

# **⚠️ Important notes:**

* If you are restoring from a **previous database backup** (.sql), **import** it before visiting the site.
* If you are starting **fresh**, Drupal installer will automatically guide you.
* **Composer install is absolutely mandatory** after cloning — otherwise nothing will work.

# **🚀 Bonus Tip**

If you want **even faster setup** later, you could add a simple README.md to your GitHub project explaining these steps for yourself or your team.  
 Something like:

# How to Install

- git clone ...

- composer install

- cp web/sites/default/default.settings.php web/sites/default/settings.php

- mkdir web/sites/default/files

- Setup DB

- Run PHP server

**You Said:**

ok show me how to export a backup of my db and restore the db from backup.

**Response:**

# **📚 Part 1: Export (Backup) Your Drupal Database**

Suppose your Drupal database is named drupal9db.

### **🛠 Command to Export**

1. mysqldump -u root -p drupal9db > drupal9db\_backup.sql

✅ This does:

* -u root = use MySQL root user
* -p = ask for password
* drupal9db = your database name
* > = save output into a file
* drupal9db\_backup.sql = backup file created

# **📦 Where will the backup file go?**

* The .sql file will be created in **your current terminal folder**.
* You can move it to safe place, push to private storage, etc.

# **📚 Part 2: Restore (Import) Your Drupal Database**

Suppose you **pulled** a project and want to **rebuild the database** from the backup file.

### **🛠 Command to Restore**

First, create an empty database:

1. mysql -u root -p

Inside MySQL:

1. CREATE DATABASE drupal9db;
2. EXIT;

✅ Then import the .sql file into the database:

1. mysql -u root -p drupal9db < drupal9db\_backup.sql

✅ This does:

* -u root = using root user
* -p = ask for password
* drupal9db = database where you want to import
* < = load file into database
* drupal9db\_backup.sql = backup file

# **⚡ Full Quick Summary**

|  |  |
| --- | --- |
| **Task** | **Command** |
| Backup (Export) | mysqldump -u root -p drupal9db > drupal9db\_backup.sql |
| Create empty DB (optional) | CREATE DATABASE drupal9db; |
| Restore (Import) | mysql -u root -p drupal9db < drupal9db\_backup.sql |

# **🧠 Important Best Practices**

* **Always backup before updating modules**, updating Drupal, or making big changes.
* **Never push .sql files to public GitHub** (they can contain sensitive user data!).
* **Name backups with date/time** like:

drupal9db\_backup\_20240513.sql

# **🛡️ Bonus Tip: Backup and compress**

You can even **compress** backup to make it smaller:

1. mysqldump -u root -p drupal9db | gzip > drupal9db\_backup.sql.gz

And restore compressed backup like:

1. gunzip < drupal9db\_backup.sql.gz | mysql -u root -p drupal9db

✅ Saves space, faster uploads.

# **✅ Now you can:**

* Export database easily
* Restore database easily
* Move your project around fully!