



HOWTO: Installing and Running WordPress on Ubuntu GNU/Linux

9 May 2007

Step right this way for instructions on setting up your own instance of WordPress using Apache 2 (with mod rewrite), PHP5, and MySQL. I'm using Ubuntu 6.10 (Edgy Eft), but I suspect this will apply to 7.04 (Feisty Fawn) also.

You may already know that WordPress is free blogging software, Apache is a free web server, PHP is a free server-side scripting language, and MySQL is a free database, but maybe like me you haven't used the supporting "AMP" pieces that much and would like to get WordPress running with a minimum of fuss and bother.

NOTE: This guide is for running WordPress locally for development purposes. It may help you in setting up a WordPress instance to be served on the Internet, but you'll need to look elsewhere for security advice. **(See further disclamation below.)**

Let's light the (GNU) LAMP!

Install

You can probably use the Synaptic Package Manager for a GUI-based install of the necessary components, but I think it's easier to use `apt-get`.

First you need to make sure you have access to the right repositories, and for this I like the GUI. Try:

System » Administration » Synaptic Package Manager, and then

Settings » Repositories. You might see something like this:



(Checked options: universe, main, source code.)

My install worked with these options; no need for *encumbered* software as far as I know.

Psst! Make sure you close the windows for any graphical package managers before running `apt-get` below, or you could get:

```
E: Could not get lock /var/lib/dpkg/lock - open (11 Resource temporarily unavailable)
E: Unable to lock the administration directory (/var/lib/dpkg/), is another process using it?
```

Notes about supporting actors...

We're going to use Apache 2 in this guide. I've used 1.x versions before and 2.x seems easier to work with (once I figured out a few things!). (2.0, specifically, since that's what `apt-get` gave me.) If you already have Apache 1.3 installed and running, I'm not sure what kind of conflicts you might see. You may be comfortable with getting the older version to work. Or if you're not using it, you might try removing it (`sudo apt-get remove apache`) before installing Apache 2. (It's up to you; this isn't about the man telling you what to do.)

We'll also use phpMyAdmin as a web-based front-end to MySQL. There are other administrative tools out there but this is what I have available at my web host, so it's easier for me to use one tool and keep my personal TCO down.

OK, here goes!

```
sudo apt-get install apache2 php5 mysql-server phpmyadmin
```


That's just about everything in one command (with WordPress notably missing — we'll get to it in a bit).

As I work through this on my laptop, I see `apt-get` suggests and recommends several other packages, but so far I haven't seen where I need any of them. It will probably take a few minutes to download and install the 100+ MB of software you just asked for. Eventually you should see MySQL and Apache being started and then the install will finish soon after. (I hope you don't get any errors. I've installed on two machines now without a hiccup. Well, the first time I uninstalled and reinstalled several times without unpleasant side effects.)

How's our web server and database?

Point your web browser to **`http://localhost`**. If all went well, you should see something like this:

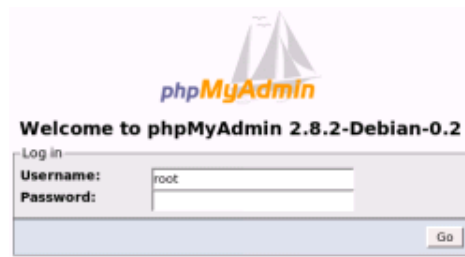
Index of /

Name	Last modified	Size	Description
 apache2-default/	27-Sep-2006 11:41	-	
 phpmyadmin/	28-Sep-2006 10:12	-	

Apache/2.0.55 (Ubuntu) PHP/5.1.6 Server at localhost Port 80

Let's verify that Apache and phpMyAdmin (and therefore MySQL and PHP) are ok before moving on to WordPress. Clicking on the **apache2-default** link should bring up the comforting site of the default Apache installation page. It says: "If you can see this, it means that the installation of the Apache web server software on this system was successful. You may now add content to this directory and replace this page." That and the "Powered by APACHE" logo are always welcome when trying to use this stuff.

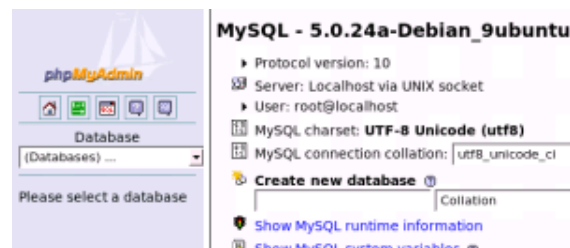
Go back and click on the **phpmyadmin** link, to see the login page:



phpMyAdmin Login

Don't Panic! If you have problems opening the phpmyadmin link, for example if your browser is Firefox and it complains that it doesn't know what to do with a PHTML file, please see my "briar patch" notes below.

The MySQL root user does not have a password by default. Enter **root** as the username and click on **Go**. You should see the initial view after logging in. We'll set passwords later.



phpMyAdmin Initial View

Installing this way, both Apache and MySQL were set up to start when the machine starts. I haven't started figuring out how system boot stuff is handled (init / startup services / what-have-you), but I learned that things are changing with the init process starting from Ubuntu 6.10/Edgy, to something called Upstart.

Next, let's find WordPress a home so we can move from installing to configuring.

WordPress!

We could have installed WordPress via the same call to apt-get earlier, but I prefer to handle this one manually. The latest version as I write this is 2.1.3, so let's download it from wordpress.org.

(Or you can get older versions from the WordPress Release Archive. For myself, I'm sticking with the 2.0 series for now and hoping they'll sort out some of the database category/tag/link concerns in 2.2. As far as this guide goes, 2.0 and 2.1 are essentially the same.)

(Or! You can copy an existing WordPress installation and start from there.)

Let's say our username is **rocky**, and the name of our blog is "**Mind Grapes**," for which we have the domain name **mindgrapes.net**. We'll download `wordpress-2.1.3.tar.gz` and save it to `/home/rocky/Desktop`.

And let's say we want to run the blog from `/home/rocky/web/mindgrapes/www`:

```
mkdir -p ~/web/mindgrapes/
```

(-p creates parent directories if needed. Don't worry; we'll get to the www in a moment.)

```
tar -xvf ~/Desktop/wordpress-2.1.3.tar.gz -C ~/web/mindgrapes/
```

(Extract contents of the tar file to directory mindgrapes. And be verbose about it!)

You'll see a scrolling list of files as they're extracted to the newly created directory **wordpress** under mindgrapes. Then:

```
cd ~/web/mindgrapes
mv wordpress www
```

(You are of course free to use wordpress as your dirname and free to put this wherever you want. I use www because it seems appropriate for the root of my web site, and I like to have it in my home directory so it will get backed up with my other stuff. Later we'll address some permissions issues with this setup.)

Under www, you'll see the WordPress subdirectories wp-admin, wp-content, wp-includes, and miscellaneous php files and so on.

OK! We're installed.

Configure

Apache 2

I spent the most time getting Apache configured correctly. I even resorted to reading some of the Apache *documentation*, believe it or not. It didn't take so long to get a single site defined, but I had a heckuva time setting up two. At the moment I have two WordPress sites I want to work with locally, so I slogged ahead until I got it.

You're probably going to have to restart Apache many times in this process. Here are two ways you can do this:

```
sudo /usr/sbin/apache2ctl restart
```

or

```
sudo /etc/init.d/apache2 restart
```

(/usr/sbin is on the path by default in Ubuntu so you don't have to type it to run programs in that directory. If you run apache2, without /etc/init.d, you'll get a different program that is in /usr/sbin with perhaps unexpected results.)

You can also supply stop or start instead of restart with either command. apache2ctl seems to be quieter by nature, while init.d/apache2 will report things like:

```
* Forcing reload of apache 2.0 web server... [ ok ]
* Stopping apache 2.0 web server... [ ok ]
* Starting apache 2.0 web server... [ ok ]
```

(Beware of a gotcha if you try running these commands without sudo. See "Briar Patch" below.)

Running multiple web sites (or even just one) gets us in to the topic of virtual hosts (vhosts), and the choice of using name-based versus IP-based virtual hosts. The Apache docs recommend name-based virtual hosts unless there is a specific reason to use IP-based, and explains some of those reasons. For our local web server scenario, name-based hosts will work fine.

Our area of interest is /etc/apache2. Apache 2 uses the sites-available and sites-enabled directories here for easier management of multiple sites.

Sites-available contains config files for the different web sites. **Sites-enabled** contains symbolic links that point to files in sites-available, so it's easy to enable and disable individual sites by creating and removing the symlinks.

To accommodate multiple web sites (virtual hosts), we'll first edit a key configuration file, `virtual.conf`. (You'll probably have to create it.)

```
sudo vi /etc/apache2/conf.d/virtual.conf
```

You just need this one setting:

```
#  
# enable multiple virtual hosts  
#  
NameVirtualHost *
```

(Thanks to the Debian Administration site for this crucial information! I wasn't having any luck with multiple sites until I put this together with some other settings in the individual site config files. See "Briar Patch" below for misadventures in this area.)

Now let's go to `/etc/apache2/sites-available`, where there is probably just a single config file for the default localhost site that we looked at earlier. Make a backup copy of the file named **default**, calling it (e.g.) **default.bck**. (Using `sudo` for all this stuff, since root owns the files.)

Open up the default file for editing. The first two lines are:

```
NameVirtualHost *  
<VirtualHost *>
```

Delete or comment out the first line to get:

```
#NameVirtualHost *  
<VirtualHost *>
```

Then look for:

```
<Directory />  
    Options FollowSymLinks  
    AllowOverride None  
</Directory>  
<Directory /var/www/>  
    Options Indexes FollowSymLinks MultiViews  
    AllowOverride None  
    Order allow,deny  
    allow from all  
    # Uncomment this directive is you want to see apache2's  
    # default start page (in /apache2-default) when you go to /  
    #RedirectMatch ^/$ /apache2-default/  
</Directory>
```

The Apache web site has a nice page explaining these settings. You can have multiple **Directory** sections that apply rules (directives) in order from shortest match to longer, so that `<Directory />` would go first. (This looks odd to me, since that's the form of an empty XML tag, but it starts the section.) Let's lock things down a bit more for the base:

```
<Directory />  
    Options FollowSymLinks  
    AllowOverride None  
    Order deny,allow  
    Deny from all  
</Directory>
```

And now configure the access to the default web site root to be more strict also:

```
<Directory /var/www/>
  Options Indexes FollowSymLinks MultiViews
  AllowOverride None
  Order deny,allow
  Deny from all
  Allow from 127.0.0.1
</Directory>
```

That makes the default site accessible only from the local machine. You can specify another machine (e.g. with IP address 192.168.1.50) on your local network to also have access like so:

```
allow from 127.0.0.1 192.168.1.50
```

(There is also a <Directory> cgi-bin section where you can do the same things, or you can remove that whole section if you're not doing cgi-bin stuff.)

That's it for "default." Save the file, and make a copy of it, naming the new file **local.mindgrapes.net**.

Before we edit our new mindgrapes config file, let's test our changes. Try restarting Apache using one of the methods above, and see if you can still reach the default web page (<http://localhost/apache2-default/>). I don't remember this from the first install on my desktop, but on my second installation I got this message when restarting:

```
apache2: Could not determine the server's fully qualified domain name,
using 127.0.1.1 for ServerName
```

It doesn't seem to cause any problems. We could experiment with defining "ServerName" in the default config, but I didn't have to do this previously. I'm not going to worry about it right now. Explanations from readers are welcome. :-)

Now:

```
sudo vi /etc/apache2/sites-available/local.mindgrapes.net
```

We're going to make several changes. **Add** a new line above the ServerAdmin setting:

```
ServerName local.mindgrapes.net:80
```

Change the DocumentRoot setting to:

```
DocumentRoot /home/rocky/web/mindgrapes/www
```

(No trailing slash on the DocumentRoot.)

Modify the <Directory /var/www/> section for mindgrapes:

```
<Directory /home/rocky/web/mindgrapes/www/>
  Options Indexes FollowSymLinks MultiViews
  AllowOverride FileInfo
  Order deny,allow
  Deny from all
  Allow from 127.0.0.1
</Directory>
```

AllowOverride has to do with the **.htaccess** file. Allowing FileInfo enables us to use mod_rewrite for URL manipulation (“pretty” permalinks). It appears to work ok for basic .htaccess stuff. I haven’t worked that much with it yet but if problems you can try AllowOverride All. We’ll cover mod_rewrite in a bit.

Change the ErrorLog setting to:

```
ErrorLog /var/log/apache2/error_mindgrapes.log
```

Change the CustomLog setting to:

```
CustomLog /var/log/apache2/access_mindgrapes.net combined
```

Enabling/Disabling Sites

We’re done with the config file for **local.mindgrapes.net**. Save the file and return to your command line. There are two ways to enable the site. One way is to manually create the symbolic link in the sites-enabled dir:

```
cd /etc/apache2/sites-enabled  
sudo ln -s /etc/apache2/sites-available/local.mindgrapes.net local.mindg
```

That’s it. To disable the site, just delete the link. Alternatively, there are utility programs named /usr/sbin/a2ensite and /usr/sbin/a2dissite. (Make sure to run these as *sudo* or you may have confusing results!)

Run `sudo a2ensite` from any directory, and it will ask you which site you want to enable, listing your choices from the sites-available directory. After picking one, it will prompt that you need to restart apache2 to enable. (You can see the symlink created in sites-enabled right away.) You could also run `sudo a2ensite local.mindgrapes.net` to avoid the interactive prompt.

Run `sudo a2dissite` to disable a site (remove its symlink). Or supply the site name on the command line to again avoid interactivity.

There are no dependencies between these techniques. You can create a link manually and then delete it with a2dissite or create it with a2ensite and delete it manually. At first when I learned about these utils I thought it would be easier to create and delete the links manually and not worry about storing the extra **a2*** information in my brain, but as I closed in on a thousand attempts at getting things working, I started to see their value. They’re easy to use and it’s nice to be able to run them from wherever you happen to be.

One thing I’m not sure how to do with a2ensite is make the link be a different name than the config file. For example, notice how the default sites-enabled link is named **000-default**? The sites are started in alphanumeric order and it can be important which ones start first (or before later problem ones, maybe). This is more of a concern for “real” web servers with lots of sites, but it’s something to be aware of.

Making it “real”

However you enable it, once you have the link for **local.mindgrapes.net** in place, restart Apache. Now there’s just one more thing we need to do so we can visit our local web page:

```
sudo vi /etc/hosts
```

The hosts file is a place where you can define names that belong to the local machine and name-to-IP-address translations for machines on your local network. (I use it for my network since I haven’t gotten around to figuring out local DNS stuff yet.)

Add a line to tell Apache that our mindgrapes site exists locally:

```
127.0.0.1    local.mindgrapes.net
```

Or you can add it to an existing definition for 127.0.0.1:

```
127.0.0.1    localhost local.mindgrapes.net.
```

(I prefer using separate lines.)

Now! We can finally try visiting our local site! Try `http://local.mindgrapes.net` in your web browser. If all is well, you should see a WordPress page pointing out that it can't find the `wp-config.php` file. Or if that file exists, there's a good chance you'll see the famous WordPress "Error establishing a database connection" page. Either is a fine site to behold since it shows that we've successfully configured Apache for our web site.



Error establishing a database connection

This either means that the username and password inform `wp-config.php` file is incorrect or we can't contact the data
This could mean your host's database server is down.

Apache mod_rewrite

We can run WordPress with the current Apache configuration, but there's one more thing we'll want to use (or that *I* want to use, anyway) that we might as well configure now and get it over with.

Apache modules are handled just like sites, with `mods-available` and `mods-enabled` directories. Looking in `/etc/apache2/mods-enabled`, I see:

```
cgi.load
php5.conf
php5.load
userdir.conf
userdir.load
```

I didn't look before installing PHP, but I'm guessing the `php5` files were added earlier by our `apt-get` install. I'll further guess that `userdir` is what enables individual users to create public web sites in `~/public_html`. We want to see `rewrite.load` in here also:

```
sudo ln -s /etc/apache2/mods-available/rewrite.load rewrite.load
```

(There are also the `a2enmod` and `a2dismod` utilities similar to the ones for sites. Run them with `sudo`.)

This is the rewrite engine that allows you to customize permalinks in WordPress, for example using `/2007/05/05/descriptive-post-title/` instead of `/?p=123`. If you don't want to use this feature, you don't have to install `mod_rewrite`. (Note that it also allows you to set up other useful kinds of URL manipulation in your `.htaccess`.)

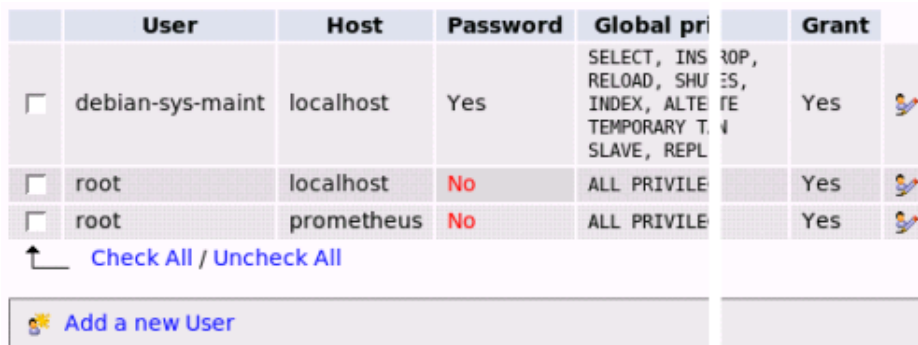
Restart apache one last time to fully enable `mod_rewrite`.

Notice that we didn't need to edit `/etc/apache2/apache2.conf` (unless we saw the PHTML problem described below in "Briar Patch") or `/etc/apache2/httpd.conf` at all. You may be familiar with **`httpd.conf`** if you've noodled around with Apache before. It's here for compatibility purposes, but not needed in Apache 2. **`apache2.conf`** has a bunch of stuff in it, but thanks to the nice new design of Apache 2, we don't have to monkey with it for our simple setup.

MySQL

This one is much easier than the Apache configuration. You need a database user and a database, and then you're either going to allow WordPress to create the tables for you, or you'll import an existing set of tables from another installation.

Go to the <http://localhost/phpmyadmin/> web page. Remember that the root user doesn't have a password yet. Enter **root** as the user and click on the **Go** button. Find and click on a link that says **Privileges**. You should see something like this:



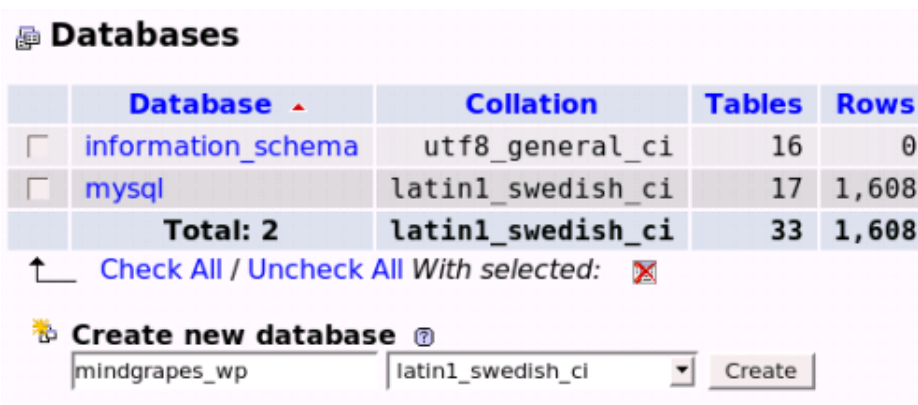
	User	Host	Password	Global privileges	Grant	
<input type="checkbox"/>	debian-sys-maint	localhost	Yes	SELECT, INSERT, UPDATE, RELOAD, SHUTDOWN, INDEX, ALTER, TEMPORARY TABLES, SLAVE, REPLICATION	Yes	
<input type="checkbox"/>	root	localhost	No	ALL PRIVILEGES	Yes	
<input type="checkbox"/>	root	prometheus	No	ALL PRIVILEGES	Yes	

[Check All / Uncheck All](#)

[Add a new User](#)

I don't know what the "debian-sys-maint" user is for nor what its password is. I also don't know why there are two root users, one for localhost and one for the machine name. You can set passwords for the two root user entries by clicking on the edit icon on the far right. (Looks like a person and a pen.) In the resulting screen, there'll be several sections, one of which is **Change Password**. Click the appropriate option button, enter the password twice, and click on the **Go** button that goes with that section.

Following instructions from wordpress.org, we'll now create our database. From the main page where you found **Privileges**, you should see another link for **Databases**. (Or you might see a **Databases** tab from whatever screen you're on. Also note that you can return to the main page by clicking on a **Server: localhost** link found at the top of many pages.) Your Databases page might look a lot like:



Databases


	Database	Collation	Tables	Rows
<input type="checkbox"/>	information_schema	utf8_general_ci	16	0
<input type="checkbox"/>	mysql	latin1_swedish_ci	17	1,608
	Total: 2	latin1_swedish_ci	33	1,608

[Check All / Uncheck All](#) With selected:

[Create new database](#)

Under **Create new database**, enter **mindgrapes_wp** and select the type of collation from the dropdown box. I have no idea what collation is. Something to do with character sets or whatever. Through past instructions or happenstance, I use **latin1_swedish_ci**. Spin the wheel and click on **Create**. You should get a results page with a SQL statement that tells you the database was created. (Note that the database is also now available in a dropdown list in the left column.)

Now we'll return to the Privileges page to create our WordPress database user. Click on the **Add a new User** link as shown in the Privileges page screenshot above. Add like so:

 **Add a new User**

Login Information

User name: mindgrapes_user

Host: localhost

Password: *****

Re-type:

Generate Password:

Global privileges ([Check All](#) / [Uncheck All](#))

User name: (Use text field:) » mindgrapes_user
 Host: (Local) » localhost
 Password: (Use text field) » ***** (And Re-type)

Leave all options unchecked under **Global Privileges** and **Resource limits** at the default of 0. Click on **Go**.

You'll get a results page with a couple of SQL statements. From this page (or from the Privileges page where you can see the User overview, and then click on the **Edit privileges** icon for mindgrapes user), look for the section **Database-specific privileges**. Select **mindgrapes_wp** (the underscore is escaped) in the database dropdown for **Add privileges on the following database:**. The screen refreshes and you might see:

 **User 'mindgrapes_user'@'localhost' - Database mindgrapes_wp : Edit Privileges**

Database-specific privileges ([Check All](#) / [Uncheck All](#))

With many checkboxes for Data, Structure, and Administration. Click on the **Check All** link, and then **Go**.

And that's about it for phpMyAdmin/MySQL, except for importing an existing database, speaking of which:

Importing/Exporting Databases, Coming Later...

I'd like to cover the importing and exporting of existing databases, but I'm going to let that go until another time.

The 2.0 versions of WordPress have a database backup feature that give you a file you can import using phpMyAdmin. Looks like 2.1 WordPress uses an XML format for exporting and importing. Does that mean you have to create a blank database before importing? I kind of liked creating the database by importing from a .sql backup file, but I suspect the new feature will work fine. I also wonder if 2.1 can import from non-XML formats? Although if you have a .sql file it will probably import in to phpMyAdmin easy enough, and then I'd hope 2.1 would upgrade the database.

You can use phpMyAdmin to export and import your database. If you're wanting to take an existing live database and start your development blog with it, I think the instructions below will still prove useful for setting everything else up. In my initial install of 2.0 WordPress, it was simple to import the backup .sql file in

phpMyAdmin.

I hate to cop out on this part, but I want to get this initial version posted. This is free and open source documentation: post early and often! Details may show up soon, but they may not.

WordPress

First we have to configure WordPress to talk to the database. Go to your `~/web/mindgrapes/www` directory. If you don't have a `wp-config.php` file, create one by copying `wp-config-sample.php`. Open it for editing and you will see:

```
// ** MySQL settings ** //
define('DB_NAME', 'wordpress'); // The name of the database
define('DB_USER', 'username'); // Your MySQL username
define('DB_PASSWORD', 'password'); // ...and password
define('DB_HOST', 'localhost'); // 99% chance you won't need to change
this value
```

Enter in your values from the previous steps for `DB_NAME`, `DB_USER`, and `DB_PASSWORD`. `DB_HOST` stays "localhost".

If you want to have a single file that you can use on your local and production WordPress instances, try this:

```
if ($_SERVER['HTTP_HOST'] == 'local.mindgrapes.net')
{
    //four definitions from above
}
else
{
    //four definitions from above
}
```

And then enter the appropriate values for each server.

After saving `wp-config.php`, try visiting `http://local.mindgrapes.net` with your browser. Hopefully you'll see the WordPress logo and the message:

It doesn't look like you've installed WP yet. Try running `install.php`.

And **install.php** is a link. Click on it! (Or try `http://local.mindgrapes.net/install.php` directly.) You'll get a page saying "Welcome to WordPress Installation." Now you can click on the **First Step** link. Enter a title (e.g. "Mind Grapes") and an email (e.g. "a@b.com"). I'd uncheck the option to have your blog appear in search engines if you're using this as a local development site.

Continue to Second Step...

That's it! You are given a username **admin** and a random password. You can proceed to **wp-login.php**, enter in your credentials, and you're in.

Pretty Permalinks

If you want to use Apache `mod_rewrite` for customized permalinks, you'll need to create a `.htaccess` file in your web site root folder. (In our case, `~/web/mindgrapes/www/.htaccess`) You can do this manually, or WordPress can write to it if you if you make it writable for the Unix user that runs WordPress. (More on permissions stuff soon.) Let's consider both ways.

With no `.htaccess` file in place, I tried **Options » Permalinks » Date and name based permalinks** and clicked the **Update Permalink Structure** button and was rewarded with:

```
Warning: Cannot modify header information - headers already sent by
(output started at
/home/rocky/web/mindgrapes/www/wp-admin/admin-header.php:16) in
/home/rocky/web/mindgrapes/www/wp-includes/functions.php on line 1221
```

Are you sure you want to change your permalink structure to: ?

No Yes

That was disturbing, but I clicked **Yes** and it came back with a page that suggested it worked, mostly, although it had this to say:

If your `.htaccess` file were writable, we could do this automatically, but it isn't so these are the mod rewrite rules you should have in your `.htaccess` file. Click in the field and press CTRL + a to select all.

And in a text box for my copying and pasting pleasure:

```
# BEGIN WordPress
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteBase /
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.php [L]
</IfModule>
# END WordPress
```

(If you had installed in a different folder than the root of your web site, this would be different: there'd be `RewriteBase /blog/` and `RewriteRule . /blog/index.php [L]`.)

So I guess we would have had to create the file before **Update Permalink Structure**, and then `chmod 666 .htaccess`. It's just as easy to:

```
vi ~/web/mindgrapes/www/.htaccess
```

And paste the lines in that way. That should be all you need — try a new post or use the “Hello World!” starter post to see if you're in business with customized permalinks. In my first install this worked right away. On the second, I ran in to some issues.

If you have trouble here, don't panic. Look at it as a learning opportunity. Review the instructions and your configuration and you should be able to get it. (Unless I gave you bad instructions. Please let me know if so and I'll correct them.)

Other useful things you can do with `.htaccess` that are more relevant to your live site:

- Make it so that you always use `www` (or not) in your URL, so that if people visit, `http://mindgrapes.net`, it will come out as `http://www.mindgrapes.net`, or vice versa.
- Prevent people from “hot-linking” to your image files.

But this post is much too long already to get in to these topics.

That's all, folks. I hope at this point you have a functioning local version of WordPress. Read on for some discussion of file ownership, permissions, and file locations.

File Stuff

(This isn't very thorough — it assumes you are familiar with how groups and permissions work. It's just meant to give you some ideas on what you can do with organizing your site files.)

One question I had after installing this is, "Who runs this stuff? Which Unix user is behind WordPress?" I'm pretty sure it's the user that runs Apache, and on Ubuntu that seems to be the `www-data` user.

This may be important for you depending on what plugins you run. Your web site files are all in `~/web/mindgrapes/www` and are owned by the "rocky" user. Let's say you're using Arne Brachhold's excellent Google Sitemap Generator. It wants to write files named `sitemap.xml` and `sitemap.xml.gz` to your blog's root folder. You can create those files as empty files and make them writable by the "world:"

```
touch ~/web/mindgrapes/www/sitemap.xml sitemap.xml.gz
chmod 666 sitemap.xml*
```

And then the plugin can overwrite them. But I preferred to put myself in the `www-data` group and make the group for the files be `www-data` with corresponding group write permissions. I haven't started setting up the even more excellent WP-Cache 2.0 plugin by Ricardo Galli, but I suspect it will work well to make the cache folder writable by the `www-data` group. (Note: In both of my machines where I've installed Apache, the `www-data` user doesn't appear in the GUI groups manager. I had to use the `addgroup` command.)

Another thing I wanted to do was keep my MySQL database files in my home directory so they'd get backed up with my nightly backup. I looked in `/etc/mysql/my.cnf` to see that the `datadir` was pointing to `/var/lib/mysql`. In there is a `mindgrapes_wp` directory owned and only readable by the `mysql` user, full of `*.frm`, `*.MYD`, and `*.MYI` files. I added myself to the `mysql` group (this one was available in the GUI) and then:

```
sudo /etc/init.d/mysql stop
sudo mv /var/lib/mysql/mindgrapes_wp ~/web/mindgrapes/mysql-db
sudo ln -s ~/web/mindgrapes/mysql-db mindgrapes_wp
cd ~/web/mindgrapes
sudo chown rocky:mysql mysql-db
chmod 770 mysql-db
sudo /etc/init.d/mysql start
```

Now the MySQL directory is under `mindgrapes` and at the same level as the `www` folder. That appeals to me for keeping things grouped. Seems to work just fine this way.

This stuff with the groups is something I'd only do locally. Even then it may not be a great idea. If the "rocky" user account were compromised, the attacker could cause additional mischief. I'd appreciate any comments from more experienced readers about the pros and cons of this kind of thing, or if there are other options that would have served me better.

Briar Patch

Please, Br'er Fox, please don't throw me in the briar patch!

PHTML Files

When I first installed this stuff on my desktop machine and tried:

```
http://localhost/phpmyadmin and
http://localhost/phpmyadmin/index.php
```

I'd get a dialog window from Firefox:

```
You have chosen to open  
[blank line]  
which is a: PHTML file
```

And a prompt for an application to use for opening it. I picked `/usr/bin/firefox`. The browser refreshed and then prompted again. This time I cleverly selected to always use this application, and was rewarded with a cascading opening of new tabs until Firefox crashed.

Searching in Google I found a lot of people with this problem. One main theme in the advice was to clear out the Firefox cache. (**Tools » Clear Private Data** is one way you can do that.) I tried that and restarted Firefox and tried a few more times for good measure, but no luck.

A lot of other pages suggested dabbling in `httpd.conf` or `apache2.conf`. I finally tried:

```
sudo vi /etc/apache2/apache2.conf
```

And followed advice to uncomment these lines:

```
#AddType application/x-httpd-php .php  
#AddType application/x-httpd-php-source .phps
```

And add `.phtml`:

```
AddType application/x-httpd-php .php .phtml  
AddType application/x-httpd-php-source .phps
```

Then a restart of Apache (see instructions in the Apache section above), and maybe a final clearing of the cache in Firefox, and finally I could open phpMyAdmin in the browser.

(I didn't have this problem when I installed on my 6.10/Edgy laptop. Those lines are still commented there. And then I couldn't get it to break again on my desktop so I could write about the details here. Kind of a flaky problem.)

Trying to stop/start/restart Apache as a regular user

If you try issuing commands to Apache as a regular user (most likely because you forgot to use `sudo`), you'll get errors that might not make your mistake obvious. Especially with `/usr/sbin/apache2ctl`:

```
httpd not running, trying to start  
(13): make_sock: could not bind to address [::]:80  
no listening sockets available, shutting down  
Unable to open logs
```

With `/etc/init.d/apache2`, a little bit more of a clue with the first line:

```
open: Permission denied  
* Forcing reload of apache 2.0 web server...  
httpd (pid 27117?) not running
```

And it hangs there until you try CTRL+C.

Permalink / mod_rewrite

I had some problems with this on my second install. I hadn't set AllowOverride in the Apache local.mindgrapes.net config file. My posts appeared on the front page, with a customized permalink, but clicking on the post titles gave me (e.g.) Not Found / The requested URL /2007/05/07/test-post/ was not found on this server.

If you're getting errors like this, keep checking and tweaking your settings in your config file. (And make sure mod_rewrite is enabled and that Apache has been restarted!)

Apache "has no VirtualHosts" errors

Until I found a way to use the virtual.conf folder in conjunction with the individual site files, I had **a lot** of errors like these when starting Apache:

```
[Sun Apr 22 15:23:00 2007] [warn] NameVirtualHost *:0 has no
VirtualHosts
[Sun Apr 22 15:23:41 2007] [warn] NameVirtualHost *:80 has no
VirtualHosts
```

Found lots of other people out there struggling with this. (Although not so many while I mistakenly searched for "has no virtual hosts" for a while.) Lots of discussion and advice about how Apache can be really picky about things matching, and that whether you specify port 80 or some other port or a star (*) or whatever, things need to match, so I tried lots of combinations of these settings in the sites-available config:

```
NameVirtualHost local.movingtofreedom.org:80
<VirtualHost local.movingtofreedom.org:80>
ServerName local.movingtofreedom.org:80
```

Or:

```
NameVirtualHost *
<VirtualHost *>
ServerName *
```

And so on. I could get a single site running, but never multiple sites correctly. As soon as I added a second site, again with the "NameVirtualHost yada-yada has no VirtualHosts". One or another site might work — if I had default and the intended blog set up, both URLs would lead to the same site. Finally discovered as shown above that it works to put NameVirtualHost * in the virtual.conf file and make the individual site files start with <VirtualHost *>.

Disclaimer

a caveat (or two or three)

(Along with the usual disclaimer.)

I use Surpass Hosting for movingtofreedom.org and have been very happy with them. My \$6/month service is almost always up and the web site is usually responsive. I'm happy to outsource my web hosting this way. For now, at least, I wouldn't want to maintain my own web server facing the Internet.

But I do like having a local development instance of my site; a sandbox for writing posts, experimenting, and making changes before moving things to "production." About the time I started this blog, I found some excellent instructions for using WAMP to run WordPress in Windows. (Also has a good explanation there of reasons to use a test blog.) I knew that all the component

pieces were easily available in GNU/Linux, but it seemed like a big step to take at the time.

Now that I finally have a working backup system in place, it's time to start moving things over and doing actual work in GNU. Time to migrate my local WordPress to Ubuntu so I can start writing about free software on a *free* operating system. (I started writing this post in Windows and am now editing and finishing it in Ubuntu.)

The *caveat*: since my intention is to only use this installation of WordPress locally, I'll pay little attention to security concerns related to running WordPress, MySQL, PHP, and Apache on a server that is exposed directly to the Internet. If that's your plan, please use other sources to learn proper security. Related is that I'm only using the supporting components for WordPress, so my instructions are mostly WordPress-centric and don't account for other uses.

So you should be warned that this is a rather **narrow-minded** guide.

And, of course, this write-up is a reflection of the blunderings of a newbie. I got it to work, and I hope I can help you do the same, but I am most certainly not an expert. I have only a weak grasp of why things work the way they do. Like many people, I'm an empirical user: I experiment with things and rely on random web searches to get them working, rather than reading and understanding all of the documentation. I've discovered a few incantations that produced a working version of a cool free software program, and even replicated them on a second machine. I probably could've just as easily rendered my whole system inoperable. Well, I have more confidence that, but in any case: *Caveat emptor!*

Finally!

THE END. Good luck, and please let me know if the install worked out for you, if there are any errors, or if things could be made clearer. Thank you for your patronage.

Posted by **Scott Carpenter** on 9 May 2007 at 9:20 pm
filed under how-to, ubuntu, wordpress

Comments

1. **Moving to Freedom: Ubuntu and Wordpress Installation...**

Many of you know Scott Carpenter who blogs at Moving to Freedom and is an accomplished writer for the Free Software Magazine. (He also hangs out here once in awhile, as well.)

I just went to visit Scott's blog and I am seriously impressed. He just p...

Posted by OpenSourceCommunity.org on 10 May 2007 at 8:18 am

2. `<Directory />` would go first. (This looks odd to me, since that's the form of an empty XML tag, but it starts the section.)

This has nothing at all to do with xml :-) the "/" is the top or root directory of your file system (like in 'cd /'). Apache has the ability to access every part of your file system, so in order to deny access to things you haven't specified explicitly, the Apache folks has started with denying access to everything. It's a security feature. Also, if you don't have this, Apache will look for .htaccess files from the root directory downwards to the website directory instead of just from the website directory (<http://httpd.apache.org/docs/2.2/howto/htaccess.html>).

Posted by BjarneDM on 19 May 2007 at 3:01 am

3. debian-sys-maint

This is an account used to do a sanity check on your mysql installation on startup of the mysql server. I've been messing around with Ubuntu at a previous job, and if I do remember correctly, you'll find that /etc/init.d/mysql does some clever things to your databases when you start the server.

Some additional information here:

<http://forums.debian.net/viewtopic.php?t=14543>

<http://jeremy.sunriseroad.net/2007/03/correct-a-broken-debian-sys-maint-account/>

Posted by BjarneDM on 19 May 2007 at 3:12 am

4. Hi, Bjarne! Thank you for the information and pointers.

For the `<Directory>` settings, I was unclear if that slash was the root of the whole file system or the root starting from "DocumentRoot." Looking at that htaccess howto, I see that it appears to be the whole file system, and as you mentioned, Apache will check all those parent directories.

But then I wonder how you'd access those other directories above DocumentRoot from the browser? If I make my default site have AllowOverrideAll set to all, and try `http://localhost` (or `http://localhost/apache2-default/`), I'm starting at /var/www (I think). I tried `http://localhost/..` and didn't go any higher. If you can't get to them, it doesn't make sense to me to have a way of checking .htaccess in them.

Finally, I did understand that `<Directory />` was pointing to the root of *something*, but I was wondering how it gets away with looking like an empty XML tag, but now I realize the obvious — this isn't an XML file at all and is probably not parsed as one. (I could tell that, but since it used the convention of `<TAG </TAG` to open and close the Directory and VirtualHost sections, I must have been confused.) :-)

(Now I should update the guide to further emphasize the importance of the /Directory section.)

Posted by Scott Carpenter on 19 May 2007 at 7:53 am

5. Hi Scott,

Glad you gave me your card yesterday (5/18) at the open-source-group (TSOUE?) meeting at MPLS CC. I've spent the last hour knocking around your

site and have really enjoyed it. I've put holds on a couple of books that you recommended (Stephenson, Stallman) and have browsed over the abundance of helpful info you've made available here.

This is a great resource. One that I've bookmarked.

Jim

Posted by Jim on 19 May 2007 at 11:20 am

6. Hi, Jim! Thanks for stopping by — I'm glad that you're enjoying the site. Let's keep in touch and I hope to hear some stories about future success in using MySQL in place of Oracle. :-)

Posted by Scott Carpenter on 19 May 2007 at 11:36 am

7. Glad to have been of help :-)

Well, then maybe the <Directory /> section is more of a performance thing than a security thing. But the way it's described by the Apache people I do get the impression it's !both! a security and a performance thing. I did try to comment out the <Directory /> section on my own test server and was unable to do a directory traversal. At least on IIS this has been of great concern and the basis for a lot of security vulnerabilities.

<http://httpd.apache.org/docs/2.2/urlmapping.html>

Posted by BjarneDM on 19 May 2007 at 11:48 am

8. Hi,
Thanks for your efforts.

I recently decided to move and update my wordpress instance to a new server, with a new domain name. Here is what worked for me.

<http://www.madteckhead.com/blog/2007/05/25/wordpress-22-server-migration/>

N

Posted by madteckhead on 25 May 2007 at 8:26 am

<http://www.movingtofreedom.org/2007/05/09/how-to-wordpress-on-ubuntu-gnu-linux/>

Except where otherwise noted, this work is licensed under a
Creative Commons Attribution-ShareAlike 3.0 License.
<http://creativecommons.org/licenses/by-sa/3.0>