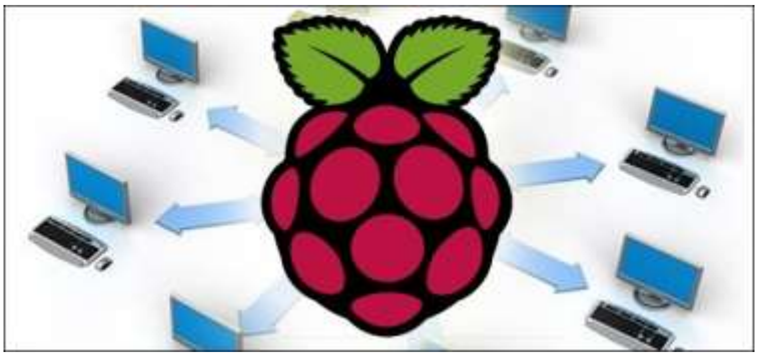
How to Turn a Raspberry Pi into a Low-Power Network Storage Device



Mix together one Raspberry Pi and a sprinkle of cheap external hard drives and you have the recipe for an ultra-low-power and always-on network storage device. Read on as we show you how to set up your own Pi-based NAS.

Why Do I Want to Do This?

The benefit of having an always-on network storage device is that it's extremely convenient to have your data (or backup destination) always accessible to the computers both inside and outside your network. The downside, in most instances, is that you're consuming a fair amount of power for the convenience.

Our office server, for example, runs 24/7 and consumes almost $200 worth of power a year. A Raspberry Pi-based network storage device on the other hand, consumes about $5 worth of power per year.

We'll be the first to grant you that a full fledged server is going to have more storage space and the capability to do more work (such as transcoding a multi-terabyte video collection in a reasonable span of time). For most people, however, the principle purpose of having an always-on computer somewhere in the house is to serve as a file server and file backup repository. For such tasks the Raspberry Pi is more than powerful enough and will save you a chunk of change in power use.

What Do I Need?

This tutorial builds on our previous tutorial: The HTG Guide to Getting Started with Raspberry Pi and we'll assume you've already completed that—in other words you already have your Raspberry Pi, got it powered up, hooked to a mouse and keyboard, and you've installed Raspbian on it.

In addition to the gear you'll need from the Getting Started with Raspberry Pi tutorial, you'll only the following hardware:

One (at minimum) USB external hard drive for simple network backups and file serving

or

Two (at minimum) USB external hard drives for local data redundancy

That's it! If you just want a simple network attached drive, you'll only need one hard drive. We highly recommend using at least two hard drives in order to allow for local (at the Raspberry Pi) data redundancy. For the purposes of this tutorial we're using a matching pair of Seagate Backup Plus 1TB Portable External Hard Drives. They're super small, don’t require an external power source, and were on sale when we were shopping for parts.

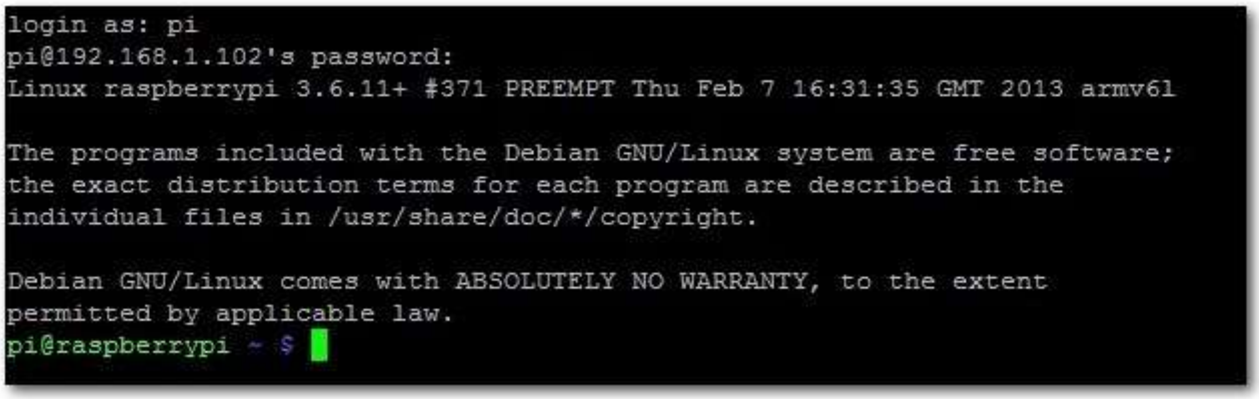
You can use any external hard drives you have on hand but it’s ideal to use small low-power drives if possible since the whole theme of the project is to set up a tiny and low-power NAS you can just tuck out of the way and forget about.

Before we continue, there are a couple design choices we made in terms of how we’re configuring our Raspberry Pi NAS that you should be aware of. While most users will want to follow along exactly as we’ve done it, you may wish to tweak specific steps to better fit your needs and how you use the computers on your network.

First, we’re using NTFS-formatted hard disks. Should the Raspberry Pi NAS fail for some reason or we want to quickly copy information over a USB 3.0 connection instead of via the network, having NTFS-formatted disks makes it dead simple to take the portable USB drives we’re using on the NAS build and plug them right into one of the many Windows machines we use every day.

Second, we’re using Samba for our network shares, again because of the convenience of meshing the Raspberry Pi NAS with our predominantly Windows network.

Preparing for and Mounting the External Hard Drives



```
login as: pi
pi@192.168.1.102's password:
Linux raspberrypi 3.6.11+ #371 PREEMPT Thu Feb 7 16:31:35 GMT 2013 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
pi@raspberrypi ~ $
```

Once you have gathered up the hardware, followed along with the Getting Started with Raspberry Pi tutorial to get up to speed (and are running Raspian) it’s time to start setting up your Pi as a NAS.

The first order of business is to hook up the hard drives to the Raspberry Pi (or the attached USB hub depending on your configuration and whether or not the hard drives are self-powered or externally powered). Once the hard drives are attached and the Pi is powered up it’s time to get working.

Note: We’re using two hard drives. If you have decided to only use one hard drive simply disregard all the commands in this section intended to mount/modify or otherwise interact with the second hard drive.

We’re going to be doing all of our work within the terminal. As such you can either work directly at your Raspberry Pi using LXTerminal in Raspian or you can SSH into your Raspberry Pi using a tool like Putty. Either way is fine.

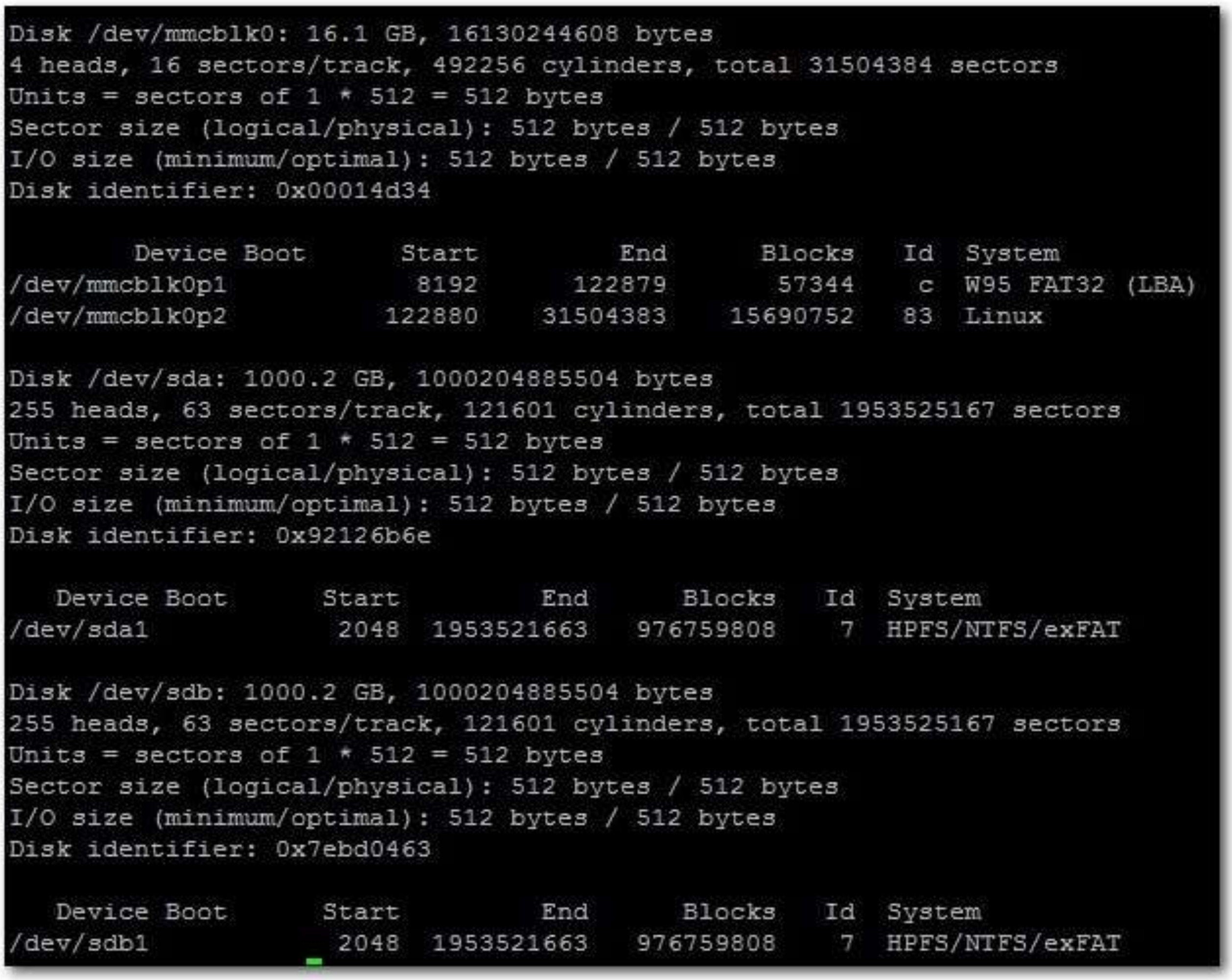
Once you’re at the command line the first thing you need to do is to add in support to Rasbian for NTFS-formatted disks. To do so type the following command:

sudo apt-get install ntfs-3g

It'll take a minute or two for the packages to download, unpack, and install. Once the NTFS package is installed it's time to look for the unmounted partitions of the attached external hard drives.

sudo fdisk -l

At minimum you should see two disks, if you've added in a secondary disk for data mirroring (as we have) you should see three like so:



```
Disk /dev/mmcblk0: 16.1 GB, 16130244608 bytes
4 heads, 16 sectors/track, 492256 cylinders, total 31504384 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00014d34

        Device Boot      Start         End      Blocks   Id  System
/dev/mmcblk0p1            8192      122879       57344    c  W95 FAT32 (LBA)
/dev/mmcblk0p2          122880    31504383    15690752   83  Linux

Disk /dev/sda: 1000.2 GB, 1000204885504 bytes
255 heads, 63 sectors/track, 121601 cylinders, total 1953525167 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x92126b6e

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048  1953521663   976759808    7  HPFS/NTFS/exFAT

Disk /dev/sdb: 1000.2 GB, 1000204885504 bytes
255 heads, 63 sectors/track, 121601 cylinders, total 1953525167 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x7ebd0463

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            2048  1953521663   976759808    7  HPFS/NTFS/exFAT
```

The first disk /dev/mmcb1k0 is the SD card inside the Raspberry Pi that houses our installation of Raspbian. We're going to leave that one completely alone.

The second disk, /dev/sda is our first 1TB external hard drive. The third disk, /dev/sdb is our second 1TB external hard disk. The actual partitions we're interested in on these two disks are /sda1/ and /sdb1/, respectively. Make a note of the hard drive names.

Before we can mount the drives, we need to create a directory to mount the drives to. For the sake of simplicity we're going to simply make directory called USBHDD1 and USBHDD2 for each drive. First we have to make the drives. At the command line enter the following commands:

```
sudo mkdir /media/USBHDD1
sudo mkdir /media/USBHDD2
```

After you've created the two directories, it's time to mount the external drives to each location. Again at the command line enter the following commands:

```
sudo mount -t auto /dev/sda1 /media/USBHDD1
sudo mount -t auto /dev/sdb1 /media/USBHDD2
```

At this point we have the two external hard drives mounted to the USBHDD1 and USBHDD2 directories, respectively. It's time to add in a specific directory to both drives to hold our shared folders (for the sake of keeping things tidy and compartmentalizing our work on the drives). Enter the following commands:

```
sudo mkdir /media/USBHDD1/shares
sudo mkdir /media/USBHDD2/shares
```

Now it's time to install Samba so we can access the storage from elsewhere on the network. At the command line enter:

```
sudo apt-get install samba samba-common-bin
```

When prompted to continue type Y and enter. Sit back and relax as everything unpacks and installs. Once the Samba package finishes installing, it's time to do a little configuration. Before we do anything else, let's make a backup copy of the Samba configuration file in case we need to revert to it. At the command line, type the following command line:

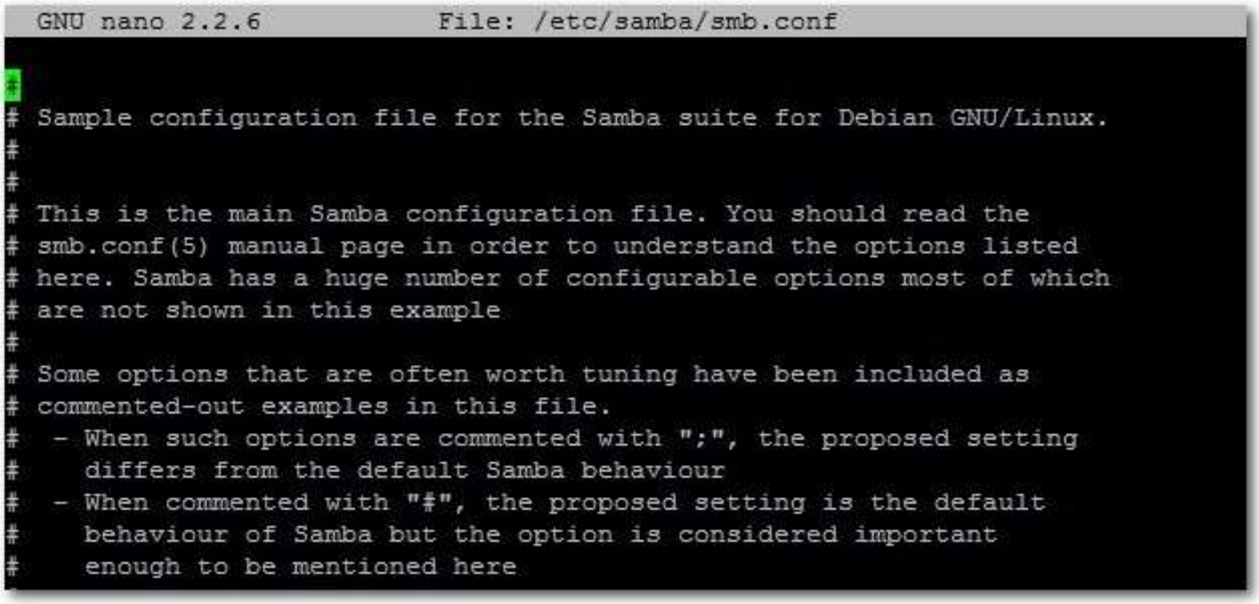
```
sudo cp /etc/samba/smb.conf /etc/samba/smb.conf.old
```

This simply creates a backup of the configuration file with the filename smb.conf.old and leaves it in the same directory as the original configuration file.

Once we've created the backup it's time to do some basic editing in the Samba config file. Type the following at the command line:

```
sudo nano /etc/samba/smb.conf
```

This will open the nano text editor and allow us to make some simple changes. If this is your first time using nano, we would strongly suggest checking out The Beginner's Guide to Nano, the Linux Command-Line Text Editor. You should see something like the following in your terminal window:



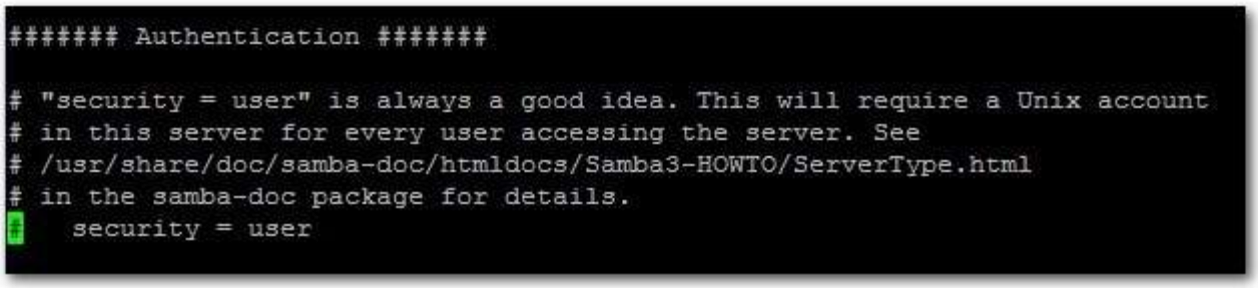
```
  GNU nano 2.2.6             File: /etc/samba/smb.conf

#
# Sample configuration file for the Samba suite for Debian GNU/Linux.
#
#
# This is the main Samba configuration file. You should read the
# smb.conf(5) manual page in order to understand the options listed
# here. Samba has a huge number of configurable options most of which
# are not shown in this example
#
# Some options that are often worth tuning have been included as
# commented-out examples in this file.
#  - When such options are commented with ";", the proposed setting
#    differs from the default Samba behaviour
#  - When commented with "#", the proposed setting is the default
#    behaviour of Samba but the option is considered important
#    enough to be mentioned here
```

Nano is completely keyboard controlled, use the arrow keys to move the cursor to the location you want to edit. As you click down through the configuration settings, you'll see a few worth making note of or changing.

The first is the workgroup identifier, by default **workgroup = WORKGROUP**. If you're using a different name for your home workgroup, go ahead and arrow over to change that now, otherwise leave it as the default.

Our next stop is to turn on user authentication for our samba storage, otherwise anyone with general access to our network (like guest Wi-Fi users) will be able to walk right in. Scroll down in the Samba config file until you get to the section that reads:



```
####### Authentication #######

# "security = user" is always a good idea. This will require a Unix account
# in this server for every user accessing the server. See
# /usr/share/doc/samba-doc/htmldocs/Samba3-HOWTO/ServerType.html
# in the samba-doc package for details.
#   security = user
```

Remove the # symbol from the security = user line (by highlighting it with the cursor and pressing delete) to enable username/password verification for the Samba shares.

Next, we're going to add an entirely new section to the configuration file. Scroll all the way down to the very bottom of the file and enter the following text:

[Backup]
comment = Backup Folder
path = /media/USBHDD1/shares
valid users = @users
force group = users
create mask = 0660
directory mask = 0771
read only = no

Note: Whatever you put in the brackets in the top line is going to be the name of the folder as it appears on the network share. If you want another name other than "Backup" now is the time to edit it.

Press CTRL+X to exit, press Y when asked if you want to keep changes and overwrite the existing configuration file. When back at the command prompt enter the following command to restart the Samba daemons:

sudo /etc/init.d/samba restart

At this point we need to add in a user that can access the Pi's samba shares. We're going to make an account with the username backups and the password backups4ever. You can make your username and password whatever you wish. To do so type the following commands:

sudo useradd backups -m -G users

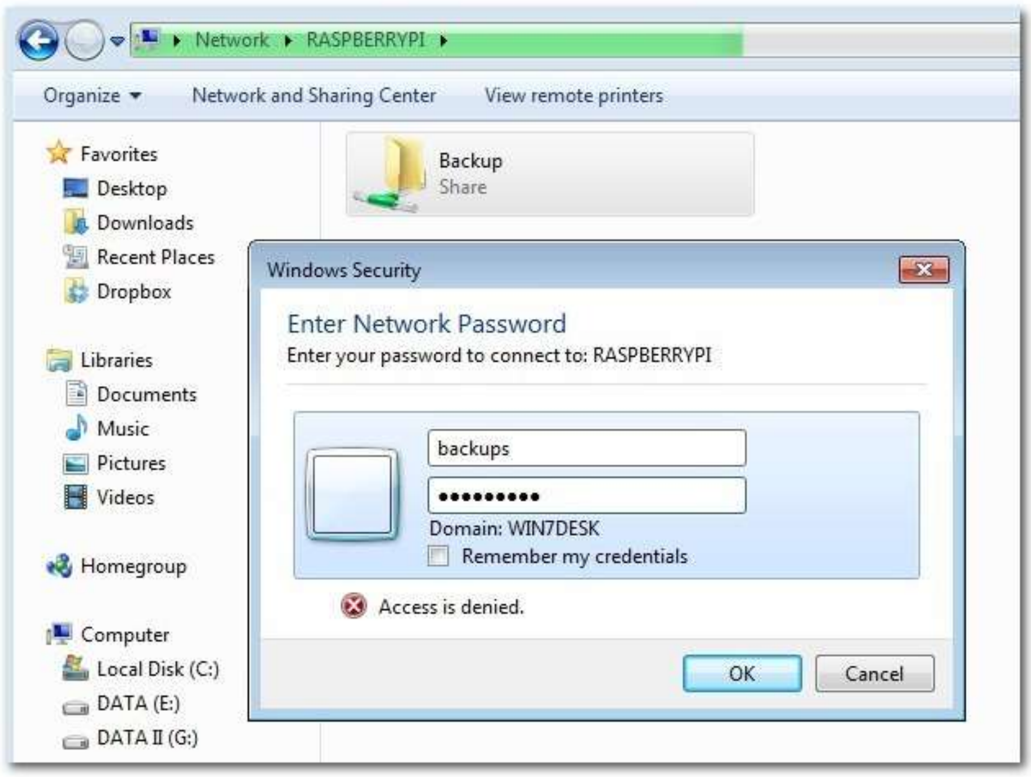
sudo passwd backups

You'll be prompted to type in the password twice to confirm. After confirming the password, it's time to add "backups" as a legitimate Samba user. Enter the following command:

sudo smbpasswd -a backups

Enter the password for the backup account when prompted. Once you have created the user account and password you do not need to restart the Samba daemon again as we've already instructed it to be on the lookout for authenticated users. We can now hop onto any Samba-capable machine on our network and test connectivity to the network share.

From a nearby windows machine we opened up the Windows file explorer, clicked on Network, confirmed that the hostname RASPBERRYPI was in the WORKGROUPS workgroup and clicked on the shared folder Backups:



Windows Security

Enter Network Password

Enter your password to connect to: RASPBERRYPI

backups

••••••••••

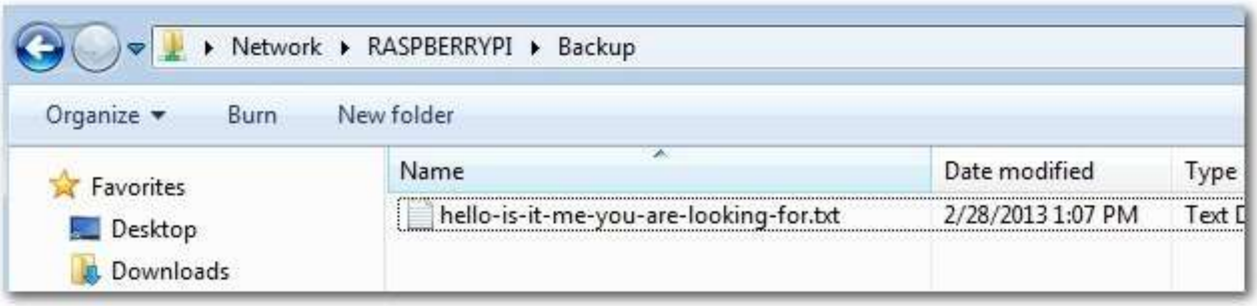
Domain: WIN7DESK

Remember my credentials

Access is denied.

When prompted, enter the credentials you created in the previous step (if you're following along line for line, the login is backups and the password is backups4ever).

Once your credentials are accepted, you will be treated to an empty folder as there isn't anything in the share yet. To double check everything is working smoothly, let's create a simple file from the computer we tested the connection with (in our case, the Windows 7 desktop). Create a txt file like so:

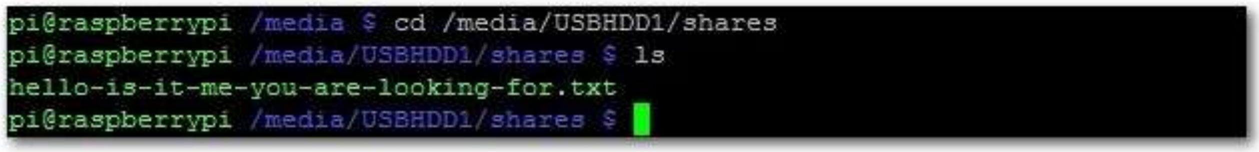


Name	Date modified	Type
hello-is-it-me-you-are-looking-for.txt	2/28/2013 1:07 PM	Text D

Now, from the command line we have been working all this time, let's check to see if the file we created on the Windows desktop appears properly within the share directory we created. At the command line type the following command:

cd /media/USBHDD1/shares

ls



```
pi@raspberrypi /media $ cd /media/USBHDD1/shares
pi@raspberrypi /media/USBHDD1/shares $ ls
hello-is-it-me-you-are-looking-for.txt
pi@raspberrypi /media/USBHDD1/shares $
```

hello-is-it-me-you-are-looking-for.txt is in the directory; our simple shared directory experiment is a success!

Before we leave this section of the tutorial, we only have one more thing to do. We need to configure our Pi so that when it restarts it will automatically mount the external hard drives. To do so we need to fire up the nano editor and make a quick edit. At the command line type:

sudo nano /etc/fstab

This will open up the file systems table in nano so we can add a few quick entries. Within the nano editor add the following lines:

/dev/sda1 /media/USBHDD1 auto noatime 0 0

/dev/sda2 /media/USBHDD2 auto noatime 0 0

Press CTRL+X to exit, press Y to save, and overwrite the existing file.

If you're only using a single hard drive for simple network sharing with no redundancy, then that's it! You're all done with the configuration process and can begin enjoying your ultra-low power NAS.

Configuring Your Raspberry Pi NAS for Simple Data Redundancy

So far our Raspberry Pi NAS is hooked up to the network, file transfer works, but there's one glaring thing missing. That secondary hard drive is configured but sitting entirely idle.

In this section of the tutorial we're going to use two simple but powerful Linux tools, rsync and cron, to configure our Raspberry Pi NAS to perform a nightly data mirror from the /shares/ folder on the primary drive to the /shares/ folder on the secondary drive. This isn't going to be a real time RAID-like data mirroring, but a daily (or semi-daily) data backup to the secondary drive is a great way to add another layer of data security.

First, we need to add rsync to our Rasbian installation. If this is your first time using rsync and you'd like to get a better overview of the command, we recommend checking out How to Use rsync to Backup Your Data on Linux.

At the command line enter the following command:

```
sudo apt-get install rsync
```

Once rsync is installed, it's time to set up a cron job to automate the process of copying files from the USBHDD1 to USBHDD2. At the command line enter the following command:

```
crontab -e
```

The command will open up your cron scheduling table in the nano text editor which should be rather familiar to you at this point in the tutorial. Go ahead and scroll down to the bottom of the document and enter the following line:

```
0 5 * * * rsync -av --delete /media/USBHDD1/shares /media/USBHDD2/shares/
```

This command specifies that every day at 5:00AM (the 0 5 part), every single day (* * *, wild cards in the year, month, day spots), we want rsync to compare the two directories, copying everything from HDD1 to HDD2 and deleting anything in the backup directory that no longer matches something in the primary directory—i.e. if we have a movie file on HDD1 we delete, we also want that file to be removed from the backup on the next synchronization.

The important part about configuring this command is that you select a time that doesn't interfere with any other network activity to the shared folders you may have scheduled. For example, if you're using your Raspberry Pi NAS as a backup destination for some sort of automated software that copies your files to the NAS at 5AM every morning, then you need to either adjust the backup time in your backup software or you need to adjust the time for the cron job on the Pi—but you can't have both the remote backup dumping data onto the network share and the Raspberry Pi trying to sync that data between local drives at the same time.

Once you've entered the crontab entry, click CTRL+X to exit and save the file. If you wish to run the rsync immediately to get the data mirrored faster and make the initial cron job a little lighter on the system, go ahead and enter the same rsync command you put into the crontab at the command line like so:

```
rsync -av --delete /media/USBHDD1/shares /media/USBHDD2/shares/
```

That's it! All you need to do at this point is check in on your Raspberry Pi in the next day or two to make sure that the scheduled job is firing off as expected and the data from /USBHDD1/shares/ is appearing in /USBHDD2/shares/.

From here on out anything you put into your Raspberry Pi-powered NAS will be mirrored daily across both hard drives.