

# LINUX VOICE

## TUTORIAL

GRAHAM MORRISON



DIFFICULTY

# ARCH LINUX: BUILD A POWERFUL, FLEXIBLE SYSTEM

Install the rolling release distro of the moment and you'll never have to wait for a package upgrade again.

Installing Arch is the Linux equivalent of base jumping. You organise yourself. Surround yourself with everything you need, stick the installation media on to a USB stick and jump. You never know how an installation is going to go until you try it, and it will always involve a bit of *ad-hoc* hacking, Googling and troubleshooting. But that's the fun of it, and that's what makes Arch different.

With Arch, you're on your own. In a world where technology is taking your personal responsibility and giving it to the cloud, or to an internet search filter or the device manufacturers,

getting your hands dirty with an operating system can be a revelation. Not only will you learn a great deal about how Linux works and what holds the whole thing together, you'll get a system you understand from the inside-out, and one that can be instantly upgraded to all the latest packages. You may also learn something about yourself in the process. And despite its reputation, it's not that difficult.

If you're a complete beginner, you may need to hold on to your hat, because installing Arch is an uncompromising adventure in core tools and functions. It's a jump into the unknown.

## 1 CREATE THE INSTALL MEDIA

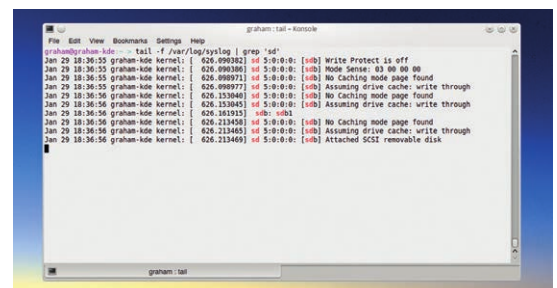
We'll start with the ISO, which you can either find on our cover DVD or download from your local Arch mirror (see <https://www.archlinux.org/download>). If you're going to install Arch onto a machine with a DVD/CD drive, you could simply burn the ISO to a blank CD, but we're going to write the ISO file to a USB thumb drive as this saves wasting a disc. You'll only need a 1GB thumb drive but this process will remove all data from the device, so make sure there's nothing on there you want to keep first.

There are many ways of transferring an ISO image to a USB drive, although copying the ISO onto the filesystem isn't one of them. Normally, our preferred method is to use the graphical tool UnetBootin, which is available for nearly all distributions, including those two alien environments, OS X and Windows. Sadly, Unetbootin won't work with Arch unless you manually edit the **syslinux.cfg** file afterwards, as this is overwritten in the transfer process. This leaves you to the mercy of **dd**, a crude command that copies the raw data from one device to another. It works, but there's no sanity checking of the output device you choose, so you have to make sure you're writing to your USB stick. If you get this wrong, you'll copy the raw bits and bytes of the Arch ISO to another storage device on your system, overwriting any data that might have been there before.

Here's our system for getting the correct device:

- 1 **sudo tail -f /var/log/syslog | grep sd**
- 2 **Clear your terminal window buffer**
- 3 **Plug in your USB drive and watch the output**

You'll see several lines appear as your system negotiates with the new USB device and, all output will



**Whenever a new USB device is connected, your system logs become a hive of activity**

include the characters 'sd'. What you need to look for is the letter that comes after 'sd', as this is the device node of the USB stick after it's connected to your system, and we need this device name for the next command, which is going to write the Arch ISO image to the USB stick. Also be aware that this device node can change, if you come back to this process after adding or removing another USB device. Here's the **dd** command for writing the ISO:

```
sudo dd bs=4M if=/path/to/arch.iso of=/dev/sdx
```

Replace the **x** in **sdx** with the letter for your device and press return. You should see the activity LED on your USB stick start to flicker as data is written. If not, press Ctrl+C immediately to stop the process and double-check everything (such as whether your USB stick has an activity LED). After the process has completed, which should only take a few moments on a modern machine, type **sync** to make sure the write buffers are flushed, and remove the stick. It's now ready to be used to install Arch.

## 2 FIRST BOOT

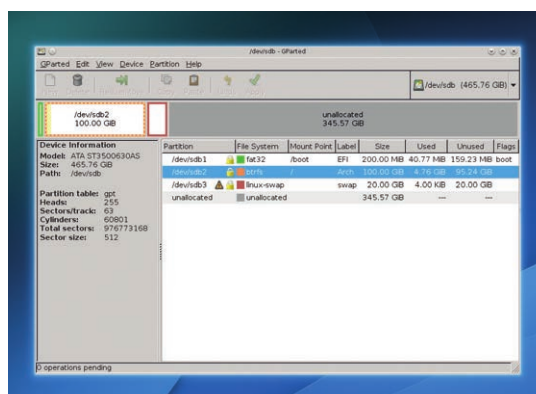
Before you plug the USB stick into the machine on which you're going to install Arch, make sure you know which hard drive you're going to use. If your machine has several drives, make a note of the capacity and model of the drive you want to use, and make sure you don't have an identical drive. If you're going to use a partition on a drive, or use up free space, we'd recommend using GParted from a live CD to set up your partitions first, or at least resize other partitions to leave enough space.

Along with a 200MB EFI partition for GUID, you'll need at least a root partition and a small swap partition. It may also help to have a separate home partition, as this makes upgrades to the root filesystem easier to handle. Most machines will boot off the USB drive by selecting the custom boot menu from your machine's boot flash screen. It's usually done by pressing the F12 key. This will present you with a list of connected drives, and you should be able to select the USB device from there. If all goes well, a moment later you'll see the Arch boot menu and you need to select the first option, 'Arch Linux archiso'.

### Networking

Your first mission is to get to the internet. We'd recommend installing the system using a wired connection if at all possible. With the system up and running, it's then much easier to configure your wireless device, but if you need to configure wireless now, check out the excellent Arch Beginners' Guide.

With a bit of luck wired internet should be working already, because Arch runs the **dhcpcd** daemon at startup, which in turn attempts to get an IP address from whatever router your kernel-configured network interface can find. Try typing **ping linuxvoice.com** to see if any packets are returned. If this doesn't work – and it didn't for us – first get the name of your



interface by typing **ip link**. It's usually the second device listed, because you should ignore the first one called **lo** (this is a system loopback device). Our PC's network device is called **enp7s0**, which you'll need to replace in the commands below. To get it working, we stop the non-functioning DHCP service, bring up the Ethernet interface, manually assign this to a valid IP address on our network and add the router as a default gateway. If you know your router's IP address, you can normally connect to its web interface to check which IP ranges are suitable for your machine, and use its IP address as the router IP address. Here are the three commands to do what we just explained – replace IP addresses to suit your own network.

```
ip link set enp7s0 up
```

```
ip addr add 192.168.1.2/24 dev enp7s0
```

```
ip route add default via 192.168.1.1
```

The final step is to type **nano /etc/resolv.conf** and add the line **nameserver 8.8.8.8** to add one of Google's nameservers to the mix. This will convert the alphanumeric URLs we normally use to the IP addressees used by the network, and you should now find that pinging a domain name works.

We used GParted to create a GPT partition scheme and a 200MB EFI partition (type ef00, labelled 'EFI'). But it might be easier to stick with old-school MBR and Grub.

### LV PRO TIP

In this tutorial we've chosen EFI booting and the GUID partitioning scheme, as this is likely to be compatible with most hardware available now, and more future proof than MBR partitioning.

## 3 FORMATTING

You should now have a fair idea at how Arch does things. It basically leaves you to do your own research and make your own decisions while creating the most common-sense environment it can. We're going to assume you've already partitioned the drive, so the first step is to make sure you know which drive to target. The best command to achieve this is **fdisk -l**. This lists all your drives, their partitions and the filesystems they're using, alongside their device nodes. Unless you've got two identical drives, you should be able to work out which one to use without too much difficulty. And if you haven't formatted your new partitions yet, they should stick out like a sore thumb. If you're only using a single drive, you'll have even fewer problems. We do know people who disconnect all other drives whilst installing Linux so

that they can be absolutely sure they won't get the wrong drive and overwrite their 500-hour *Skyrim* save position on Windows 7.

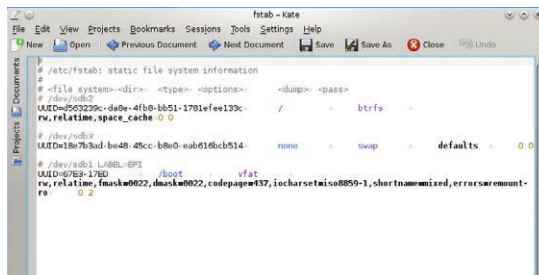
### Choose your filesystem

You should now format the partition. The safest and most sensible filesystem to use is ext4, and you can format your chosen partition by typing **mkfs.ext4 /dev/sdx2** – again, replace **x2** with your own partition. You should do this for your home partition too, and you will also want to format and define your swap partition. The command to do this is **mkswap /dev/sdx3**. You can turn this on with **swapon** followed by the device node. If you created an EFI partition yourself, rather than another OS doing this, you can format it with the command **mkfs.fat -F32 /dev/sdx**.

### LV PRO TIP

Arch's own docs are absolutely excellent. They're also very comprehensive, so don't allow them to put you off.

Our automatically generated **fstab** file didn't need any further edits



Now mount the partitions by typing:

```
mount /dev/sdx2/ /mnt
```

```
mount /dev/sdx3 /mnt/home
```

With GUID and an EFI system (rather than using the old BIOS), you'll also need to mount the EFI partition:

```
mount /dev/sdx1 /mnt/boot
```

If you're not using a separate home partition, type **mkdir /mnt/home** to create a home folder in the root partition. These are the fragile beginnings of your Arch installation. We're going to make more of an impact with the next command:

```
pacstrap -i /mnt base
```

This command installs a basic Arch system to your drive. We leave the installer at its default settings so it can grab and install all the default packages, and you'll

be left with all the packages you need. However, unlike with other distributions, that doesn't mean it's actually usable for anything yet. Following the Arch Beginners' Guide, we'll next create the **fstab** file, as this tells the distribution where to find its dependent filesystems. In the old days, we'd use labels to represent partitions, but labels can be changed or duplicated and break an **fstab** file, so we now use UUIDs. These are basically hashes derived from partition data, so Arch should never get confused unless something changes with the partition scheme. The correct file with the correct mount points and UUIDs can be generated automatically by typing:

```
genfstab -U -p /mnt >> /mnt/etc/fstab
```

You can see that this file is created in your new root filesystem, and as the file was generated automatically, you should check it's not complete insanity (try **cat /mnt/etc/fstab**). It will show your mounted filesystem along with the EFI partition we mounted on **/boot** – this should be formatted and listed as **vfat**, as per our formatting command earlier. With all that set up, we're now going to teleport ourselves into the new Arch system using 'chroot' with the following command:

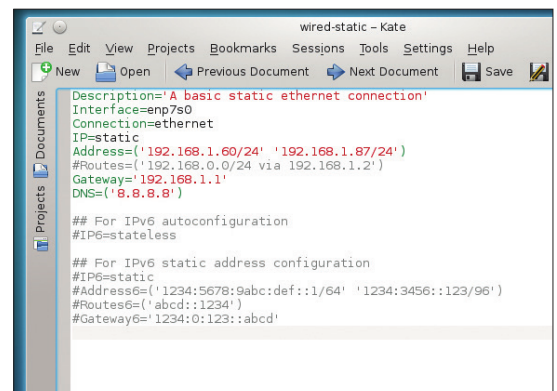
```
arch-chroot /mnt /usr/bin/bash
```

## 4 POST-CONFIG

How does it look inside your new Arch installation? Not that different than from the USB stick, except for now you're executing code from your hard drive. There's obviously lots we can do here, but we're mostly interested in getting the system up and running as quickly as possible. It's worth giving your machine a hostname, which can be done with a command like **echo linuxvoice > /etc/hostname**. Networking too should be solved in exactly the same way we got networking working earlier. If DHCP worked, just type **systemctl enable dhcpd.service** to make the required link to get it running at boot.

### Enable network profiles

An alternative to this generic solution, which didn't work for us, is to enable network profiles, such as the ones mainstream distributions use to quickly switch between network settings. First copy the **/etc/netctl/examples/ethernet-dhcp** file to **/etc/netctl/** directory, open your new file with Nano and change the device from **eth0** to whatever your machine uses (take a look at the output from **ip link**), then enable the connection for your next boot with **netctl enable ethernet-dhcp**. If you want to do the same with a static IP address, use the static Ethernet example configuration. But for this, you have to make sure DHCP isn't running when the system starts. To remove it, and any other service you no longer require, the command is **systemctl disable dhcpd.service**. Arch now uses systemd, which is why this syntax may look unfamiliar. You can check the service isn't started automatically by typing



We had to create a static networking configuration file and remove the DHCP service to get networking working.

**systemctl | grep dhcp** when you next boot. If you want netctl to automatically bring up a connection for your interface, whether you've configured it for a static or dynamic connection, type the following, but replace **enp7s0** with the name of your device:

```
systemctl enable netctl-auto@enp7s0.service
```

Before leaving the chroot environment, set a password by typing **passwd**, then **exit** and **reboot**.

We've now got to the state where we've got enough installed and configured that we can finally breathe some native life into our distribution. But before we can reboot, we need to install a bootloader. If you've already got Linux installed, or you're sharing an installation with Windows, you'll need to be careful. Installing a bootloader over a part of the disk used by

### PRO TIP

Despite updates being easy to apply on the command line, it's always worth checking that nothing requires your intervention before you do the upgrade. The best way we've found to stay in touch is to peruse Arch's Twitter account: [@archlinux](https://twitter.com/archlinux).

another operating system will stop that other operating system from booting. If you've dedicated a new single drive to Arch, which is what we'd recommend, you can install the bootloader onto this drive only – whether that's old-school MBR or newer GUID. This way, you won't break anything; your drive will boot if it's the first boot device, and it will boot if you use your system's BIOS boot menu and select an alternative drive. If you want to add your Arch installation to another Grub installation, you'll need to boot into that system and re-generate the configuration – many distributions, such as Ubuntu, can do this with a minimal of effort.

## Install a bootloader

As we're using a modern system with EFI and GUID partitioning, we're going to install a simple EFI bootloader rather than the more commonly used Grub. If you are using older partition, however, Grub can be installed with the following two command after changing **sdx** to your device:

```
pacman -S grub
```

```
grub-install --target=i386-pc --recheck /dev/sdx
```

For EFI systems, type **pacman -S gummiboot** to install the EFI bootloader package, and **gummiboot**

**install** to run the simple setup procedure. It will fail if an EFI-compatible partition can't be found, or isn't mounted. If that happens, you should install Grub.

The only other step to getting **gummiboot** to work is to create a simple configuration file called **/boot/loader/entries/arch.conf**. It should contain the following information:

<b>title</b>	<b>Arch Linux</b>
<b>linux</b>	<b>/vmlinuz-linux</b>
<b>initrd</b>	<b>/initramfs-linux.img</b>
<b>options</b>	<b>root=/dev/sda2 rw</b>

Replace the **sda2** part with the device node for your root partition and your new system should work. If it doesn't (and we don't want to be negative, but this is Arch we're talking about), the great thing about the Arch USB installer is that you can easily use it to troubleshoot your installation using the skills you've already learnt. Just reboot from the USB stick, mount the drive and **chroot** into your new Arch installation. Many serious problems can be solved this way, and it's much quicker than using a live CD. Remember this as you type **exit** to quit the **chroot** environment and **reboot** to restart your machine, because if your new Arch installation doesn't appear, you'll need to boot again from the USB stick and check the configuration,

### LV PRO TIP

Pacman is Arch's package manager, and is relatively straightforward to use. **-S** will search for and install packages; **-Ss** will search for package names and their descriptions; **-R** will remove them and **-Syu** will perform a system upgrade.

## 5 BUILD YOUR OWN HOME

You now need to log in as root, and you should check that networking is working. If not, you need to go through the same steps we went through with the USB installer.

At its most basic level, Arch is now installed and ready for you to sculpt into your perfect distribution. There are many ways to do this – you may even want to remain on the command line, but we're going to assume you'll want a graphical environment and your hardware working. Xorg, the graphical display server, can be installed with the following command:


```
pacman -S xorg-server xorg-server-utils xorg-xinit xterm mesa
```

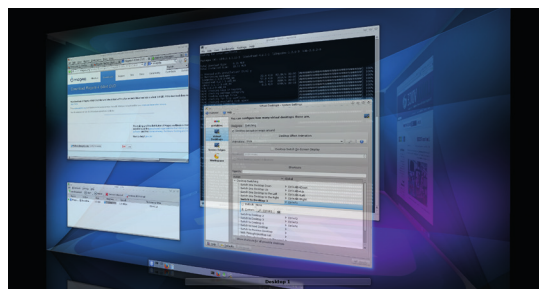
As long as you're happy using open source drivers for your graphics hardware, this is all you need for a working X session. Many of the open source drivers are good enough for desktop work, and only lack 3D performance. A simple test to make sure all this auto configuration is going to work is to type **startx** to bring up the most basic of X sessions. Unfortunately for us, it didn't work and we got a 'no screens found' error. This is probably because our screen is rubbish and isn't communicating its capabilities back to the graphics hardware. The solution is to create your own X.org config file. We're using Nvidia hardware and are happy to use Nvidia's proprietary drivers. The drivers for any modern Nvidia GPU can be installed by simply typing **pacman -S nvidia**, and rebooting your system. Nvidia's drivers are also better at detecting displays, so it might be worth trying **startx** again to see if anything has changed. You can quit the X environment by exiting all of the terminal sessions.

With X running, it's now time to install a graphical environment. Obviously this is a contentious issue, but here's the basic procedure. KDE, for example, can be installed by typing:

```
pacman -S kde-meta
```

Meta packages encapsulate other package collections, so you can fine-tune your installation. A basic KDE installation can be accomplished by grabbing the **kde-base** package, for example. **kde-meta** on the other hand downloads over 700MB of data and installs over 2GB from 558 packages. It takes a while. For Gnome, **gnome-shell** contains the basics, **gnome** has the desktop environment and the applications, while **gnome-extra** contains all the tools.

The final steps to Arch nirvana are to create a new user with **useradd -m graham**, give them a password with **passwd graham** and then to launch the KDE/ Gnome login manager by typing **kdm** or **gdm**. You'll get a fully functional login and desktop. But as you'll soon discover, this is only the end of the very beginning. With Arch, you've only just got started. 



This being Arch, you don't have to install KDE. But when was the last time you saw a gratuitous screenshot of the desktop cube looking so good?