List of Tables

Installing Ubuntu 20.04 “Focal Fossa” For arm64

We are delighted that you have decided to try Ubuntu, and are sure that you will find that Ubuntu’s GNU/Linux distribution is unique. Ubuntu brings together high-quality free software from around the world, integrating it into a coherent whole. We believe that you will find that the result is truly more than the sum of the parts.

We understand that many of you want to install Ubuntu without reading this manual, and the Ubuntu installer is designed to make this possible. If you don’t have time to read the whole Installation Guide right now, we recommend that you read the Installation Howto, which will walk you through the basic installation process, and links to the manual for more advanced topics or for when things go wrong. The Installation Howto can be found in Appendix A.

With that said, we hope that you have the time to read most of this manual, and doing so will lead to a more informed and likely more successful installation experience.

Chapter 1. Welcome to Ubuntu

This chapter provides an overview of the Ubuntu Project, and the Debian Project upon which it is based. If you already know about the Ubuntu Project's history and the Ubuntu distribution, feel free to skip to the next chapter.

1.1. What is Ubuntu?

Ubuntu is a complete Linux operating system, freely available with both community and professional support. The Ubuntu community is built on the ideas enshrined in the Ubuntu Manifesto: that software should be available free of charge, that software tools should be usable by people in their local language and despite any disabilities, and that people should have the freedom to customize and alter their software in whatever way they see fit.

- *Ubuntu will always be free of charge*, and there is no extra fee for the "enterprise edition", we make our very best work available to everyone on the same Free terms.
- Ubuntu includes the *very best in translations and accessibility infrastructure* that the Free Software community has to offer, to make Ubuntu usable by as many people as possible.
- Ubuntu is shipped in stable and regular release cycles; *a new release will be shipped every six months*. Every two even years an Ubuntu long term support (LTS) release will become available, that is supported for 5 years. The Ubuntu releases in between (known as development or non-LTS releases) are supported for 9 month each.
- Ubuntu is entirely committed to the principles of open source software development; we encourage people to use open source software, improve it and pass it on.

Ubuntu is suitable for both desktop and server use. The current Ubuntu release supports Intel x86 (IBM-compatible PC), AMD64 (x86-64), ARMv7, ARMv8 (ARM64), IBM POWER8/POWER9 (ppc64el), IBM Z zEC12/zEC13/z14 and IBM LinuxONE Rockhopper I+II/Emporer I+II (s390x).

Ubuntu includes thousands of pieces of software, starting with the Linux kernel version 5.4 and GNOME 3.28, and covering every standard desktop application from word processing and spreadsheet applications to internet access applications, web server software, email software, programming languages and tools and of course several games.

1.1.1. Sponsorship by Canonical

The Ubuntu Project is sponsored by Canonical Ltd (http://www.canonical.com/). Canonical will not charge licence fees for Ubuntu, now or at any stage in the future. Canonical's business model is to provide technical support and professional services related to Ubuntu. We encourage more companies also to offer support for Ubuntu, and will list those that do on the Support pages of this web site.

1.2. What is Debian?

Debian is an all-volunteer organization dedicated to developing free software and promoting the ideals of the Free Software community. The Debian Project began in 1993, when Ian Murdock issued an open invitation to software developers to contribute to a complete and coherent software distribution based on the relatively new Linux kernel. That relatively small band of dedicated enthusiasts, originally funded by the Free Software Foundation (http://www.fsf.org/) and influenced by the GNU (http://www.gnu.org/gnu/the-gnu-project.html) philosophy, has grown over the years into an organization of around 1026 *Debian Developers*.

Debian Developers are involved in a variety of activities, including Web (http://www.debian.org/) and FTP (ftp://ftp.debian.org/) site administration, graphic design, legal analysis of software licenses, writing documentation, and, of course, maintaining software packages.

In the interest of communicating our philosophy and attracting developers who believe in the principles that Debian stands for, the Debian Project has published a number of documents that outline our values and serve as guides to what it means to be a Debian Developer:

- The Debian Social Contract (http://www.debian.org/social_contract) is a statement of Debian's commitments to the Free Software Community. Anyone who agrees to abide to the Social Contract may become a maintainer (http://www.debian.org/doc/maint-guide/). Any maintainer can introduce new software into Debian — provided that the software meets our criteria for being free, and the package follows our quality standards.
- The Debian Free Software Guidelines (http://www.debian.org/social_contract#guidelines) are a clear and concise statement of Debian's criteria for free software. The DFSG is a very influential document in the Free Software Movement, and was the foundation of the The Open Source Definition (http://opensource.org/osd).
- The Debian Policy Manual (http://www.debian.org/doc/debian-policy/) is an extensive specification of the Debian Project's standards of quality.

Debian developers are also involved in a number of other projects; some specific to Debian, others involving some or all of the Linux community. Some examples include:

- The Linux Standard Base (http://www.linuxbase.org/) (LSB) is a project aimed at standardizing the basic GNU/Linux system, which will enable third-party software and hardware developers to easily design programs and device drivers for Linux-in-general, rather than for a specific GNU/Linux distribution.
- The Filesystem Hierarchy Standard (http://www.pathname.com/fhs/) (FHS) is an effort to standardize the layout of the Linux file system. The FHS will allow software developers to concentrate their efforts on designing programs, without having to worry about how the package will be installed in different GNU/Linux distributions.
- Debian Jr. (http://www.debian.org/devel/debian-jr/) is an internal project, aimed at making sure Debian has something to offer to our youngest users.

For more general information about Debian, see the Debian FAQ (http://www.debian.org/doc/FAQ/).

1.2.1. Ubuntu and Debian

Ubuntu and Debian are distinct but parallel and closely linked systems. The Ubuntu project seeks to complement the Debian project in the following areas:

1.2.1.1. Package selection

Ubuntu does not provide security updates and professional support for every package available in the open source world, but selects a complete set of packages making up a solid and comprehensive system and provides support for that set of packages.

For users that want access to every known package, Ubuntu provides a "universe" component (set of packages) where users of Ubuntu systems install the latest version of any package that is not in the supported set. Most of the packages in Ubuntu universe are also in Debian, although there are other sources for universe too. See the Ubuntu Components page for more detail on the structure of the Ubuntu web distribution.

1.2.1.2. Releases

Ubuntu makes a release every six months, and supports those releases for 18 months with daily security fixes and patches to critical bugs.

As Ubuntu prepares for release, we “freeze” a snapshot of Debian’s development archive (“sid”). We start from “sid” in order to give ourselves the freedom to make our own decisions with regard to release management, independent of Debian’s release-in-preparation. This is necessary because our release criteria are very different from Debian’s.

As a simple example, a package might be excluded from Debian “testing” due to a build failure on any of the 11 architectures supported by Debian “sarge”, but it is still suitable for Ubuntu if it builds and works on only three of them. A package will also be prevented from entering Debian “testing” if it has release-critical bugs according to Debian criteria, but a bug which is release-critical for Debian may not be as important for Ubuntu.

As a community, we choose places to diverge from Debian in ways that minimize the difference between Debian and Ubuntu. For example, we usually choose to update to the very latest version of Gnome rather than the older version in Debian, and we might do the same for key other pieces of infrastructure such as X or GCC. Those decisions are listed as Feature Goals for that release, and we work as a community to make sure that they are in place before the release happens.

1.2.1.3. Development community

Many Ubuntu developers are also recognized members of the Debian community. They continue to stay active in contributing to Debian both in the course of their work on Ubuntu and directly in Debian.

When Ubuntu developers fix bugs that are also present in Debian packages -- and since the projects are linked, this happens often -- they send their bugfixes to the Debian developers responsible for that package in Debian and record the patch URL in the Debian bug system. The long term goal of that work is to ensure that patches made by the full-time Ubuntu team members are immediately also included in Debian packages where the Debian maintainer likes the work.

In Ubuntu, team members can make a change to any package, even if it is one maintained by someone else. Once you are an Ubuntu maintainer it’s encouraged that you fix problems you encounter, although we also encourage polite discussions between people with an interest in a given package to improve cooperation and reduce friction between maintainers.

1.2.1.4. Freedom and Philosophy

Debian and Ubuntu are grounded on the same free software philosophy. Both groups are explicitly committed to building an operating system of free software.

Differences between the groups lie in their treatment of non-computer applications (like documentation, fonts and binary firmware) and non-free software. Debian distributes a small amount of non-free software from their Internet servers. Ubuntu will also distribute binary drivers in the "restricted" component on its Internet servers but will not distribute any other software applications that do not meet its own Ubuntu Licensing Guidelines.

1.2.1.5. Ubuntu and other Debian derivatives

There are many other distributions that also share the same basic infrastructure (package and archive format). Ubuntu is distinguished from them in a number of ways.

First, Ubuntu contributes patches directly to Debian as bugs are fixed during the Ubuntu release process, not just when the release is actually made. With other Debian-style distributions, the source code and patches are made available in a "big bang" at release time, which makes them difficult to integrate into the upstream HEAD.

Second, Ubuntu includes a number of full-time contributors who are also Debian developers. Many of the other distributions that use Debian-style packaging do not include any active Debian contributors.

Third, Ubuntu makes much more frequent and fresher releases. Our release policy of releasing every six months is (at the time of writing :-) unique in the Linux distribution world. Ubuntu aims to provide you with a regular stable and security-supported snapshot of the best of the open source world.

1.3. What is GNU/Linux?

Linux is an operating system: a series of programs that let you interact with your computer and run other programs.

An operating system consists of various fundamental programs which are needed by your computer so that it can communicate and receive instructions from users; read and write data to hard disks, tapes, and printers; control the use of memory; and run other software. The most important part of an operating system is the kernel. In a GNU/Linux system, Linux is the kernel component. The rest of the system consists of other programs, many of which were written by or for the GNU Project. Because the Linux kernel alone does not form a working operating system, we prefer to use the term “GNU/Linux” to refer to systems that many people casually refer to as “Linux”.

Linux is modelled on the Unix operating system. From the start, Linux was designed to be a multitasking, multi-user system. These facts are enough to make Linux different from other well-known operating systems. However, Linux is even more different than you might imagine. In contrast to other operating systems, nobody owns Linux. Much of its development is done by unpaid volunteers.

Development of what later became GNU/Linux began in 1984, when the Free Software Foundation (http://www.fsf.org/) began development of a free Unix-like operating system called GNU.

The GNU Project (http://www.gnu.org/) has developed a comprehensive set of free software tools for use with Unix™ and Unix-like operating systems such as Linux. These tools enable users to perform tasks ranging from the mundane (such as copying or removing files from the system) to the arcane (such as writing and compiling programs or doing sophisticated editing in a variety of document formats).

While many groups and individuals have contributed to Linux, the largest single contributor is still the Free Software Foundation, which created not only most of the tools used in Linux, but also the philosophy and the community that made Linux possible.

tions without existing environment data. It is possible to manually set bootm_size to the new U-Boot's default value by running the command "env default bootm_size; saveenv" at the U-Boot prompt.

Another possibility to circumvent relocation-related problems is to run the command "setenv fdt_high ffffffff; setenv initrd_high 0xffffffff; saveenv" at the U-Boot prompt to completely disable the relocation of the initial ramdisk and the device-tree blob.

3.7.2. Systems with UEFI firmware

UEFI ("Unified Extensible Firmware Interface") is a new kind of system firmware that is used on many modern systems and is - among other uses - intended to replace the classic PC BIOS.

Currently most PC systems that use UEFI also have a so-called "Compatibility Support Module" (CSM) in the firmware, which provides excatly the same interfaces to an operating system as a classic PC BIOS, so that software written for the classic PC BIOS can be used unchanged. Nonetheless UEFI is intended to one day completely replace the old PC BIOS without being fully backwards-compatible and there are already a lot of systems with UEFI but without CSM.

On systems with UEFI there are a few things to take into consideration when installing an operating system. The way the firmware loads an operating system is fundamentally different between the classic BIOS (or UEFI in CSM mode) and native UEFI. One major difference is the way the harddisk partitions are recorded on the harddisk. While the classic BIOS and UEFI in CSM mode use a DOS partition table, native UEFI uses a different partitioning scheme called "GUID Partition Table" (GPT). On a single disk, for all practical purposes only one of the two can be used and in case of a multi-boot setup with different operating systems on one disk, all of them must therefore use the same type of partition table. Booting from a disk with GPT is only possible in native UEFI mode, but using GPT becomes more and more common as hard disk sizes grow, because the classic DOS partition table cannot address disks larger than about 2 Terabytes while GPT allows for far larger disks. The other major difference between BIOS (or UEFI in CSM mode) and native UEFI is the location where boot code is stored and in which format it has to be. This means that different bootloaders are needed for each system.

The latter becomes important when booting `debian-installer` on a UEFI system with CSM because `debian-installer` checks whether it was started on a BIOS- or on a native UEFI system and installs the corresponding bootloader. Normally this simply works but there can be a problem in multi-boot environments. On some UEFI systems with CSM the default boot mode for removable devices can be different from what is actually used when booting from hard disk, so when booting the installer from a USB stick in a different mode from what is used when booting another already installed operating system from the hard disk, the wrong bootloader might be installed and the system might be unbootable after finishing the installation. When choosing the boot device from a firmware boot menu, some systems offer two seperate choices for each device, so that the user can select whether booting shall happen in CSM or in native UEFI mode.

Chapter 4. Obtaining System Installation Media

4.1. Official Ubuntu CD-ROMs

By far the easiest way to install Ubuntu is from an Official Ubuntu CD-ROM (http://releases.ubuntu.com/focal/) . You may download the CD-ROM image from an Ubuntu mirror and make your own CD, if you have a fast network connection and a CD burner. If you have an Ubuntu CD and CDs are bootable on your machine , you can skip right to Chapter 5; much effort has been expended to ensure the files most people need are there on the CD.

If your machine doesn't support CD booting, but you do have a CD or an ISO image, you can use an alternative strategy such as net boot, or manually loading the kernel from the CD to initially boot the system installer. The files you need for booting by another means are also on the CD; the Ubuntu network archive and CD folder organization are identical. So when archive file paths are given below for particular files you need for booting, look for those files in the same directories and subdirectories on your CD.

Once the installer is booted, it will be able to obtain all the other files it needs from the CD.

If you don't have a CD, then you will need to download the installer system files and place them on a a connected computer, so they can be used to find and boot the installer.

4.2. Downloading Files from Ubuntu Mirrors

To find the nearest (and thus probably the fastest) mirror, see the list of Ubuntu mirrors (http://wiki.ubuntu.com/Archive).

When downloading files from an Ubuntu mirror using FTP, be sure to download the files in *binary* mode, not text or automatic mode.

4.2.1. Where to Find Installation Images

The installation images are located on each Ubuntu mirror in the directory ubuntu/dists/focal/main/installer-arm64/current/images/ (http://ports.ubuntu.com/ubuntu-ports/dists/focal/main/installer-arm64/current/images) — the MANIFEST (http://ports.ubuntu.com/ubuntu-ports/dists/focal/main/installer-arm64/current/images/MANIFEST) lists each image and its purpose.

The HWE installation images are located on each Ubuntu mirror in the directory ubuntu/dists/focal-updates/main/installer-arm64/current/images/hwe-netboot/ (http://ports.ubuntu.com/ubuntu-ports/dists/focal-updates/main/installer-arm64/current/images/hwe-netboot/). The image will be available soon after HWE kernel is available and before second point release.

4.3. Preparing Files for TFTP Net Booting

If your machine is connected to a local area network, you may be able to boot it over the network from another machine, using TFTP. If you intend to boot the installation system from another machine, the boot files will need to be placed in specific locations on that machine, and the machine configured to support booting of your specific machine.

You need to set up a TFTP server, and for many machines a DHCP server, or RARP server, or BOOTP server.

The Reverse Address Resolution Protocol (RARP) is one way to tell your client what IP address to use for itself. Another way is to use the BOOTP protocol. BOOTP is an IP protocol that informs a computer of its IP address and where on the network to obtain a boot image. The DHCP (Dynamic Host Configuration Protocol) is a more flexible, backwards-compatible extension of BOOTP. Some systems can only be configured via DHCP.

The Trivial File Transfer Protocol (TFTP) is used to serve the boot image to the client. Theoretically, any server, on any platform, which implements these protocols, may be used. In the examples in this section, we shall provide commands for SunOS 4.x, SunOS 5.x (a.k.a. Solaris), and GNU/Linux.

4.3.1. Setting up RARP server

To set up RARP, you need to know the Ethernet address (a.k.a. the MAC address) of the client computers to be installed. If you don't know this information, you can boot into "Rescue" mode (e.g., from the rescue floppy) and use the command **`ip addr show dev eth0`**.

On a RARP server system using a Linux kernel or Solaris/SunOS, you use the **rarpd** program. You need to ensure that the Ethernet hardware address for the client is listed in the "ethers" database (either in the `/etc/ethers` file, or via NIS/NIS+) and in the "hosts" database. Then you need to start the RARP daemon. Issue the command (as root): **`/usr/sbin/rarpd -a`** on most Linux systems and SunOS 5 (Solaris 2), **`/usr/sbin/in.rarpd -a`** on some other Linux systems, or **`/usr/etc/rarpd -a`** in SunOS 4 (Solaris 1).

4.3.2. Setting up a DHCP server

One free software DHCP server is ISC **dhcpd**. For Ubuntu, the `isc-dhcp-server` package is recommended. Here is a sample configuration file for it (see `/etc/dhcp/dhcpd.conf`):

```
option domain-name "example.com";
option domain-name-servers ns1.example.com;
option subnet-mask 255.255.255.0;
default-lease-time 600;
max-lease-time 7200;
server-name "servername";

subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.200 192.168.1.253;
  option routers 192.168.1.1;
}

host clientname {
  filename "/tftpboot.img";
  server-name "servername";
  next-server servername;
  hardware ethernet 01:23:45:67:89:AB;
  fixed-address 192.168.1.90;
}
```

In this example, there is one server *servername* which performs all of the work of DHCP server, TFTP server, and network gateway. You will almost certainly need to change the domain-name options, as well as the server name and client hardware address. The *filename* option should be the name of the file which will be retrieved via TFTP.

After you have edited the **dhcpd** configuration file, restart it with **/etc/init.d/isc-dhcp-server restart**.

4.3.3. Setting up a BOOTP server

There are two BOOTP servers available for GNU/Linux. The first is CMU **bootpd**. The other is actually a DHCP server: ISC **dhcpd**. In Ubuntu these are contained in the `bootp` and `isc-dhcp-server` packages respectively.

To use CMU **bootpd**, you must first uncomment (or add) the relevant line in `/etc/inetd.conf`. On Debian or Ubuntu, you can run **update-inetd --enable bootps**, then **/etc/init.d/inetd reload** to do so. Just in case your BOOTP server does not run Debian or Ubuntu, the line in question should look like:

```
bootps  dgram  udp  wait  root  /usr/sbin/bootpd  bootpd -i -t 120
```

Now, you must create an `/etc/bootptab` file. This has the same sort of familiar and cryptic format as the good old BSD `printcap`, `termcap`, and `disktab` files. See the `bootptab` manual page for more information. For CMU **bootpd**, you will need to know the hardware (MAC) address of the client. Here is an example `/etc/bootptab`:

```
client:\
  hd=/tftpboot:\
  bf=tftpboot.img:\
  ip=192.168.1.90:\
  sm=255.255.255.0:\
  sa=192.168.1.1:\
  ha=0123456789AB:
```

You will need to change at least the "ha" option, which specifies the hardware address of the client. The "bf" option specifies the file a client should retrieve via TFTP; see Section 4.3.5 for more details.

By contrast, setting up BOOTP with ISC **dhcpd** is really easy, because it treats BOOTP clients as a moderately special case of DHCP clients. Some architectures require a complex configuration for booting clients via BOOTP. If yours is one of those, read the section Section 4.3.2. Otherwise you will probably be able to get away with simply adding the **allow bootp** directive to the configuration block for the subnet containing the client in `/etc/dhcp/dhcpd.conf`, and restart **dhcpd** with **/etc/init.d/isc-dhcp-server restart**.

4.3.4. Enabling the TFTP Server

To get the TFTP server ready to go, you should first make sure that **tftpd** is enabled.

In the case of `tftpd-hpa` there are two ways the service can be run. It can be started on demand by the system's `inetd` daemon, or it can be set up to run as an independent daemon. Which of these methods is used is selected when the package is installed and can be changed by reconfiguring the package.

> **Note:** Historically, TFTP servers used `/tftpboot` as directory to serve images from. However, Ubuntu packages may use other directories to comply with the Filesystem Hierarchy Standard (http://www.pathname.com/fhs/). For example, `tftpd-hpa` by default uses `/srv/tftp`. You may have to adjust the configuration examples in this section accordingly.

All **in.tftpd** alternatives available in Ubuntu should log TFTP requests to the system logs by default. Some of them support a **`-v`** argument to increase verbosity. It is recommended to check these log messages in case of boot problems as they are a good starting point for diagnosing the cause of errors.

4.3.5. Move TFTP Images Into Place

Next, place the TFTP boot image you need, as found in Section 4.2.1, in the **tftpd** boot image directory. You may have to make a link from that file to the file which **tftpd** will use for booting a particular client. Unfortunately, the file name is determined by the TFTP client, and there are no strong standards.

4.4. Automatic Installation

For unattended installs on multiple computers it's possible to do fully automatic installations using the Ubuntu Installer itself.

4.4.1. Automatic Installation Using the Ubuntu Installer

The Ubuntu Installer supports automating installs via preconfiguration files. A preconfiguration file can be loaded from the network or from removable media, and used to fill in answers to questions asked during the installation process.

Full documentation on preseeding including a working example that you can edit is in Appendix B.

4.4.2. Automatic Installation Using Kickstart

The Ubuntu installer supports automating installs using Kickstart files, as designed by Red Hat for use in their Anaconda installer. This method is not as flexible as the preconfiguration file method above, but it requires less knowledge of how the installer works.

This section documents only the basics, and differences between Anaconda and the Ubuntu installer. Refer to the Red Hat documentation (http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Installation_Guide/ch-kickstart2.html) for detailed instructions.

To generate a Kickstart file, install the `system-config-kickstart` package and run `system-config-kickstart`. This offers you a graphical user interface to the various options available.

Once you have a Kickstart file, you can edit it if necessary, and place it on a web, FTP, or NFS server, or copy it onto the installer's boot media. Wherever you place the file, you need to pass a parameter to the installer at boot time to tell it to use the file.

To make the installer use a Kickstart file downloaded from a web or FTP server, add ks=http://url/to/ks.cfg or ks=ftp://url/to/ks.cfg respectively to the kernel boot parameters. This requires the installer to be able to set up the network via DHCP on the first connected interface without asking any questions; you may also need to add ksdevice=eth1 or similar if the installer fails to determine the correct interface automatically.

Similarly, to make the installer use a Kickstart file on an NFS server, add ks=nfs:server:/path/to/ks.cfg to the kernel boot parameters. The method supported by Anaconda of adding a plain "ks" boot parameter to work out the location of the Kickstart file from a DHCP response is not yet supported by the Ubuntu installer.

To place a Kickstart file on a CD, you would need to remaster the ISO image to include your Kickstart file, and add ks=cdrom:/path/to/ks.cfg to the kernel boot parameters. See the manual page for mkisofs for details.

4.4.2.1. Additions

The Ubuntu installer supports a few extensions to Kickstart that were needed to support automatic installations of Ubuntu:

- The **`rootpw`** command now takes the **`--disabled`** option to disable the root password. If this is used, the initial user will be given root privileges via `sudo`.
- A new **`user`** command has been added to control the creation of the initial user:

  ```
  user joe --fullname "Joe User" --password iamjoe
  ```

  The **`--disabled`** option prevents any non-root users from being created. The **`--fullname`** option specifies the user's full name, as opposed to the Unix username. The **`--password`** option supplies the user's password, by default in the clear (in which case make sure your Kickstart file is kept confidential!); the **`--iscrypted`** option may be used to state that the password is already MD5-hashed.
- A new **`preseed`** command has been added to provide a convenient way to preseed additional items in the debconf database that are not directly accessible using the ordinary Kickstart syntax:

  ```
  preseed --owner gdm shared/default-x-display-manager select gdm
  ```

  Note that if the value contains any special characters, then the value must be quoted, as follows:

  ```
  preseed preseed/late_command string "sed -i 's/foo/bar/g' /target/etc/hosts"
  ```

  The **`--owner`** option sets the name of the package that owns the question; if omitted, it defaults to d-i, which is generally appropriate for items affecting the first stage of the installer. The three mandatory arguments are the question name, question type, and answer, in that order, just as would be supplied as input to the `debconf-set-selections` command.
- As of Ubuntu 6.10, the **`keyboard`** option takes X layout names. To use an X keyboard variant, set this option to **`layout_variant`**, with appropriate values of **`layout`** and **`variant`**. For example, **`in_guj`** selects the Gujarati variant of the Indian layout.

- You may use the **apt-install** command to install packages in `%post --nochroot` scripts (although you might also choose to generate a `%packages` section in a `%pre` script and include it using `%include`). Note that this does not work if the post-installation script is run in the chroot environment.

4.4.2.2. Missing features

As yet, the Ubuntu installer only supports a subset of Kickstart's features. The following is a brief summary of features that are known to be missing:

- LDAP, Kerberos 5, Hesiod, and Samba authentication.
- The `auth --enablecache` command to enable `nscd`.
- Upgrades. To upgrade from one Ubuntu release to another, use the facilities provided by `apt` and its frontends.
- Partitioning of multiple drives. Due to current limitations in the partition manager, it is only possible to partition a single drive.
- Using the `device` command to install extra kernel modules.
- Driver disks.
- Firewall configuration.
- Installation from an archive on a local hard disk or from an NFS archive.
- The `logvol --percent`, `--bytes-per-inode`, and `--fsoptions` options for certain kinds of detailed Logical Volume Management (LVM) configuration. (LVM configuration in general is experimentally supported as of Ubuntu 9.04; please let us know about your experiences with it.)
- Restrictions of a partition to a particular disk or device, and specifications of the starting or ending cylinder for a partition.
- Checking a partition for bad sectors.
- RAID configuration.
- Exclusions in %packages sections are no longer supported as of Ubuntu 6.10, as a casualty of other improvements. You may need to use a %post script instead to remove unnecessary packages.
- Pre-installation scripts and non-chrooted post-installation scripts may only be shell scripts; other interpreters are not available at this point in the installation.

4.4.2.3. Example

Here is an example Kickstart file that can be used as a starting point:

```
#
#Generic Kickstart template for Ubuntu
#Platform: x86 and x86-64
#

#System language
lang en_US
```

```
#Language modules to install
langsupport en_US

#System keyboard
keyboard us

#System mouse
mouse

#System timezone
timezone America/New_York

#Root password
rootpw --disabled

#Initial user (user with sudo capabilities)
user ubuntu --fullname "Ubuntu User" --password root4me2

#Reboot after installation
reboot

#Use text mode install
text

#Install OS instead of upgrade
install

#Installation media
cdrom
#nfs --server=server.com --dir=/path/to/ubuntu/
#url --url http://server.com/path/to/ubuntu/
#url --url ftp://server.com/path/to/ubuntu/

#System bootloader configuration
bootloader --location=mbr

#Clear the Master Boot Record
zerombr yes

#Partition clearing information
clearpart --all --initlabel

#Basic disk partition
part / --fstype ext4 --size 1 --grow --asprimary
part swap --size 1024
part /boot --fstype ext4 --size 256 --asprimary

#Advanced partition
#part /boot --fstype=ext4 --size=500 --asprimary
#part pv.aQcByA-UM0N-siuB-Y96L-rmd3-n6vz-NMo8Vr --grow --size=1
#volgroup vg_mygroup --pesize=4096 pv.aQcByA-UM0N-siuB-Y96L-rmd3-n6vz-NMo8Vr
#logvol / --fstype=ext4 --name=lv_root --vgname=vg_mygroup --grow --size=10240 \
--maxsize=20480
#logvol swap --name=lv_swap --vgname=vg_mygroup --grow --size=1024 --maxsize=8192

#System authorization infomation
```

```
auth  --useshadow  --enablemd5

#Network information
network --bootproto=dhcp --device=eth0

#Firewall configuration
firewall --disabled --trust=eth0 --ssh

#Do not configure the X Window System
skipx
```

Chapter 5. Booting the Installation System

5.1. Booting the Installer on 64-bit ARM

5.1.1. Console configuration

The graphical installer is not enabled on the arm64 `debian-installer` images for 20.04 so the serial console is used. The console device should be detected automatically from the firmware, but if it is not then after you boot linux from the GRUB menu you will see a “Booting Linux” message, then nothing more.

If you hit this issue you will need to set a specific console config on the kernel command line. Hit **e** for “Edit Kernel command-line” at the GRUB menu, and change

```
--- quiet
```

to

```
console=<device>,<speed>
```

e.g.

```
console=ttyAMA0,115200n8
```

. When finished hit **Control-x** to continue booting with new setting.

5.1.2. Juno Installation

Juno has UEFI so the install is straightforward. The most practical method is installing from USB stick. You need up to date firmware for USB-booting to work. Builds from http://releases.linaro.org/latest/members/arm/ after March 2015 tested OK. Consult Juno documentation on firmware updating.

Prepare a standard arm64 CD image on a USB stick. Insert it in one of the USB ports on the back. Plug a serial cable into the upper 9-pin serial port on the back. If you need networking (netboot image) plug the ethernet cable into the socket on the front of the machine.

Run a serial console at 115200, 8bit no parity, and boot the Juno. It should boot from the USB stick to a GRUB menu. The console config is not correctly detected on Juno so just hitting return will show no kernel output. Set the console to

```
console=ttyAMA0,115200n8
```

as described in (Section 5.1.1). **Control-x** to boot should show you the `debian-installer` screens, and allow you to proceed with a standard installation.

If this is the first time you're booting the system, try the default boot parameters (i.e., don't try setting parameters) and see if it works correctly. It probably will. If not, you can reboot later and look for any special parameters that inform the system about your hardware.

Information on many boot parameters can be found in the Linux BootPrompt HOWTO (http://www.tldp.org/HOWTO/BootPrompt-HOWTO.html), including tips for obscure hardware. This section contains only a sketch of the most salient parameters. Some common gotchas are included below in Section 5.4.

5.3.1. Boot console

If you are booting with a serial console, generally the kernel will autodetect this. If you have a videocard (framebuffer) and a keyboard also attached to the computer which you wish to boot via serial console, you may have to pass the **`console=`*`device`*** argument to the kernel, where *`device`* is your serial device, which is usually something like `ttyS0`.

You may need to specify parameters for the serial port, such as speed and parity, for instance **`console=ttyS0,9600n8`**; other typical speeds may be 57600 or 115200. Be sure to specify this option after "---", so that it is copied into the bootloader configuration for the installed system (if supported by the installer for the bootloader).

In order to ensure the terminal type used by the installer matches your terminal emulator, the parameter **`TERM=`*`type`*** can be added. Note that the installer only supports the following terminal types: `linux`, `bterm`, `ansi`, `vt102` and `dumb`. The default for serial console in `debian-installer` is **`vt102`**. If you are using a virtualization tool which does not provide conversion into such terminals types itself, e.g. QEMU/KVM, you can start it inside a **screen** session. That will indeed perform translation into the `screen` terminal type, which is very close to `vt102`.

5.3.2. Ubuntu Installer Parameters

The installation system recognizes a few additional boot parameters[1] which may be useful.

A number of parameters have a "short form" (or alias) that helps avoid the limitations of the kernel command line options and makes entering the parameters easier. If a parameter has a short form, it will be listed in brackets behind the (normal) long form. Examples in this manual will normally use the short form too.

debconf/priority (priority)

This parameter sets the lowest priority of messages to be displayed.

The default installation uses **`priority=high`**. This means that both high and critical priority messages are shown, but medium and low priority messages are skipped. If problems are encountered, the installer adjusts the priority as needed.

If you add **`priority=medium`** as boot parameter, you will be shown the installation menu and gain more control over the installation. When **`priority=low`** is used, all messages are shown (this is equivalent to the *expert* boot method). With **`priority=critical`**, the installation system will display only critical messages and try to do the right thing without fuss.

> **Note:** In order to get asked for a VLAN configuration during the network setup a priority of **`medium`** or **`low`** is needed.

1. With current kernels (2.6.9 or newer) you can use 32 command line options and 32 environment options. If these numbers are exceeded, the kernel will panic.

DEBIAN_FRONTEND

This boot parameter controls the type of user interface used for the installer. The current possible parameter settings are:

- **`DEBIAN_FRONTEND=noninteractive`**
- **`DEBIAN_FRONTEND=text`**
- **`DEBIAN_FRONTEND=newt`**
- **`DEBIAN_FRONTEND=gtk`**

The default frontend is **`DEBIAN_FRONTEND=newt`**. **`DEBIAN_FRONTEND=text`** may be preferable for serial console installs . Some specialized types of install media may only offer a limited selection of frontends, but the **`newt`** and **`text`** frontends are available on most default install media. On architectures that support it, the graphical installer uses the **`gtk`** frontend.

BOOT_DEBUG

Setting this boot parameter to 2 will cause the installer's boot process to be verbosely logged. Setting it to 3 makes debug shells available at strategic points in the boot process. (Exit the shells to continue the boot process.)

`BOOT_DEBUG=0`

This is the default.

`BOOT_DEBUG=1`

More verbose than usual.

`BOOT_DEBUG=2`

Lots of debugging information.

`BOOT_DEBUG=3`

Shells are run at various points in the boot process to allow detailed debugging. Exit the shell to continue the boot.

INSTALL_MEDIA_DEV

The value of the parameter is the path to the device to load the Ubuntu installer from. For example,

log_host
log_port

Causes the installer to send log messages to a remote syslog on the specified host and port as well as to a local file. If not specified, the port defaults to the standard syslog port 514.

lowmem

Can be used to force the installer to a lowmem level higher than the one the installer sets by default based on available memory. Possible values are 1 and 2. See also Section 6.3.1.1.

noshell

Prevents the installer from offering interactive shells on tty2 and tty3. Useful for unattended installations where physical security is limited.

debian-installer/framebuffer (fb)

Some architectures use the kernel framebuffer to offer installation in a number of languages. If framebuffer causes a problem on your system you can disable the feature using the parameter **`fb=false`**. Problem symptoms are error messages about bterm or bogl, a blank screen, or a freeze within a few minutes after starting the install.

debian-installer/theme (theme)

A theme determines how the user interface of the installer looks (colors, icons, etc.). What themes are available differs per frontend. Currently both the newt and gtk frontends only have a "dark" theme that was designed for visually impaired users. Set the theme by booting with **`theme=`*`dark`***.

netcfg/disable_autoconfig

By default, the `debian-installer` automatically probes for network configuration via IPv6 autoconfiguration and DHCP. If the probe succeeds, you won't have a chance to review and change the obtained settings. You can get to the manual network setup only in case the automatic configuration fails.

If you have an IPv6 router or a DHCP server on your local network, but want to avoid them because e.g. they give wrong answers, you can use the parameter **`netcfg/disable_autoconfig=true`** to prevent any automatic configuration of the network (neither v4 nor v6) and to enter the information manually.

disk-detect/dmraid/enable (dmraid)

Set to **`true`** to enable support for Serial ATA RAID (also called ATA RAID, BIOS RAID or fake RAID) disks in the installer. Note that this support is currently experimental. Additional information can be found on the Debian Installer Wiki (http://wiki.debian.org/DebianInstaller/).

preseed/url (url)

Specify the url to a preconfiguration file to download and use for automating the install. See Section 4.4.

preseed/file (file)

Specify the path to a preconfiguration file to load for automating the install. See Section 4.4.

preseed/interactive

Set to **`true`** to display questions even if they have been preseeded. Can be useful for testing or debugging a preconfiguration file. Note that this will have no effect on parameters that are passed as boot parameters, but for those a special syntax can be used. See Section B.5.2 for details.

auto-install/enable (auto)

Delay questions that are normally asked before preseeding is possible until after the network is configured. See Section B.2.3 for details about using this to automate installs.

finish-install/keep-consoles

During installations from serial or management console, the regular virtual consoles (VT1 to VT6) are normally disabled in `/etc/inittab`. Set to **`true`** to prevent this.

cdrom-detect/eject

By default, before rebooting, `debian-installer` automatically ejects the optical media used during the installation. This can be unnecessary if the system does not automatically boot off the CD. In some cases it may even be undesirable, for example if the optical drive cannot reinsert the media itself and the user is not there (since working from remote) to do it manually. Many slot loading, slim-line, and caddy style drives cannot reload media automatically.

Set to **`false`** to disable automatic ejection, and be aware that you may need to ensure that the system does not automatically boot from the optical drive after the initial installation.

base-installer/install-recommends (recommends)

By setting this option to **`false`**, the package management system will be configured to not automatically install “Recommends”, both during the installation and for the installed system. See also Section 6.3.4.

Note that this option allows to have a leaner system, but can also result in features being missing that you might normally expect to be available. You may have to manually install some of the recommended packages to obtain the full functionality you want. This option should therefore only be used by very experienced users.

debian-installer/allow_unauthenticated

By default the installer requires that repositories be authenticated using a known gpg key. Set to **`true`** to disable that authentication. **Warning: insecure, not recommended.**

rescue/enable

Set to **`true`** to enter rescue mode rather than performing a normal installation. See Section 8.7.

5.3.3. Using boot parameters to answer questions

With some exceptions, a value can be set at the boot prompt for any question asked during the installation, though this is only really useful in specific cases. General instructions how to do this can be found in Section B.2.2. Some specific examples are listed below.

debian-installer/language (language)
debian-installer/country (country)
debian-installer/locale (locale)

There are two ways to specify the language, country and locale to use for the installation and the installed system.

The first and easiest is to pass only the parameter `locale`. Language and country will then be derived from its value. You can for example use **`locale=de_CH`** to select German as language and Switzerland as country (`de_CH.UTF-8` will be set as default locale for the installed system). Limitation is that not all possible combinations of language, country and locale can be achieved this way.

The second, more flexible option is to specify `language` and `country` separately. In this case `locale` can optionally be added to specify a specific default locale for the installed system. Example: **`language=en country=DE locale=en_GB.UTF-8`**.