

Downloading Typora

```
wget -q0 - https://typora.io/linux/public-key.asc | sudo apt-key add -  
  
# add Typora's repository  
sudo add-apt-repository 'deb https://typora.io/linux ./'  
sudo apt-get update  
  
# install typora  
sudo apt-get install typora
```

Downloading the Aspartate theme for Typora

<https://theme.typora.io/theme/Aspartate/>

I downloaded the source code from <https://github.com/AntonVanke/typora-Aspartate-theme/releases/> as a zip file.

```
cd Downloads  
unzip typora-Aspartate-theme-0.1.7.zip  
mv typora-Aspartate-theme-0.1.7 ~/.config/Typora/themes  
cd ~/.config/Typora/themes/typora-Aspartate-theme-0.1.7  
mv aspartate.css ~/.config/Typora/themes
```

Changing the Maximum Width

<https://stackoverflow.com/questions/52983676/increase-the-editor-width-or-viewable-area-in-typora>

Creating images for CDs, DVDs, and flash drives

The command line has applications for burning data CDs, DVDs, flash, or Blu-ray disks. Before you can actually burn a disk, however, you first need to create an ISO image - that is, an archive file for your optical disc. ISO images usually have the file extension `.iso`. The name is taken from the ISO 9660 standard, which is the standard filesystem for managing files on CD-ROMS. ISO images can also contain a UDF filesystem used by DVDs and Blu-ray disks.

Regardless of the filesystem, you can create ISO images using one of two tools. The first is the `dd` command, which generally allows copying from any source, as well as the creation of ISO images. The command is especially useful for rescuing data on a dying hard drive.

Alternatively, you can create an ISO image with `mkisofs`, `genisoimage`, or `xorrisofs`. Which of these tools is available to you depends on your distribution. Because of licensing issues, few if any distributions still use `mkisofs`, although it is still available as part of CDRTools. However, because of licensing issues with `mkisofs`, Debian uses `genisoimage`. Also, `genisoimage` has not been updated for several years, so distributions like Ubuntu and Linux Mint use `xorrisofs`. All these tools do the same job and understand almost all of the same options. You can use any of them to back up your data automatically, and even to exclude individual files if necessary.

Converting and Copying with dd

The `dd` does far more than create ISO images. It really should be called `cc` - for convert and copy. However, because this name had already been assigned to the C compiler, the developers just chose the next letter of the alphabet when naming it.

The `dd` tool creates exact copies of media, whether they are hard disk partitions, CDs, or DVDs. Also, `dd` supports reliable, blockwise reading and writing operations. Because `dd` does not process or interpret these blocks, the underlying filesystem is not important. In fact, `dd` isn't even fazed by hard disks with errors. The basic command syntax for `dd` is:

```
dd if=<source> of=<target>
```

The `if` option tells `dd` where to read the source data (input file), and the `of` option defines the destination (output file). The source and target are often devices, such as hard disk partitions or CD/DVD drives. Alternatively, you can use an equals sign to point to a file. To copy the hard disk partition `/dev/sda1` bit for bit to `/dev/sdb1`, you could type:

```
dd if=/dev/sda1 of=/dev/sdb1
```

You can also use `dd` in the shell to copy a CD or DVD quickly. To create an ISO image, for example, use the command:

```
$ dd if=/dev/sr0 of=myimage.iso
1529961+0 records in
1529960+0 records out
783339520 bytes (783 MB) copied, 90.6944 s, 8.6 MB/s
```

When you are using `dd`, you do not need to mount the medium to perform a quick copy: Just replace the `/dev/sr0` drive designator with the device name for your drive.

The ISO image is written to a file called `myimage.iso` in the current directory. You can rename the file as necessary.

Optimizing dd Options

The `dd` tool also has a number of options. One practical option that speeds up the program considerably is `bs` (for "block size"). By default, `dd` uses 512-byte blocks; that is, it reads 512 bytes at a time and writes them to the output file. If you select a larger block size, you can speed up this process. For example, typing

```
dd if=/dev/sda1 of=/dev/sdb1 bs=2k
```

tells `dd` to copy the partition in blocks of 2KB (2048 bytes). If the last block is smaller than the specified block size, `dd` will not pad it. Padding is when bits or characters are used to fill up unused portions of a data structure.

```
$ dd if=/dev/sda1 of=/dev/sdb1 bs=6k
16059+1 records in
16059+1 records out
98670592 bytes (95 MB) copied 13.801482 s, 6.9 MB/s
```

The output tells you that `dd` has copied 16059 blocks of 6144 bytes each with one remaining block of 4096 bytes.

Besides block size, you can specify how many block `dd` reads. To copy 40MB, just write `bs=1M count=40`. The `count` option specifies the number of blocks. This makes sense if you want to save a hard disk boot sector; for example, entering

```
dd if=/dev/sda of=bootsector bs=512 count=1
```

will just copy the first 512-byte block.

Rescuing with dd

If you are faced with the daunting task of rescuing data from damaged filesystems, `dd` is essential. Before you repair the damage, you first should create a backup copy. To do so, use `dd` to create a 1:1 copy of the damaged system and then use the copy for your rescue attempt.

Because `dd` excludes defective sectors from the copy by default, you need to enable the `conv=noerror,sync` option,

```
dd bs=512 conv=noerror,sync if=/dev/sda of=/dev/sdb
```

which tells `dd` to continue reading and storing data, even if it discovers defective sectors. The `noerror` tells `dd` not to stop on errors, and `sync` pads unreadable sectors with zeros.

Generating ISO images

The `mkisofs` command used to be the most common for creating ISO images in Linux. Because of licensing issues and some conflicts within the developer community, `genisoimage` emerged as a fork of the `mkisofs` codebase. Another tool called `xorriso` offers a range of ISO creation and manipulation features. The `xorrisofs` command launches `xorriso` in `mkisofs` emulation mode. The result is that `mkisofs`, `genisoimage`, and `xorrisofs` all have very similar features and command-line options. Check your own Linux distribution to see which of these commands is available (they might not all be present). Debian, Ubuntu, and other Debian-based alternatives ship with `genisoimage`, which will be used in the following examples. The basic syntax for all three is:

```
genisoimage <parameter> -o <myfile>.iso /<directory>/<data>
```

The `-o` flag lets you define the target file name. This is followed by the data you want to store in the image. As an optional parameter, you can tell the tool to enable Rock Ridge extensions by setting the `-r` option, which is useful chiefly for enabling longer file names. To set privileges and file ownership, you could specify `-R`. The Joliet extension, which enables the support of non-Latin characters, is enabled by the `-J` flag:

```
genisoimage -J -R -o <myfile>.iso /<directory>/<data>
```

The `-V` option lets you specify a name (volume ID) for the CD/DVD. If the name includes blanks, special characters, or both, don't forget to use double quotation marks:

```
genisoimage -V "Backup, 29th of June" ...
```

For more detailed output, you could enable the `-v` (verbose) option. The opposite of the verbose option is `-quiet`. If you prefer to avoid seeing status messages in your terminal window but don't want to do without the information these messages provide, use

```
genisoimage ... -log-file log.txt ...
```

to pipe the output of the `genisoimage` command to a logfile.

Creating Backups

Genisoimage and *xorrisofs* have a number of practical options for creating regular backups. For example, the `-m` option lets you exclude files from an image. The file name arguments follow the option. *Genisoimage* can even handle wildcards and multiple names. For example, to exclude all HTML files from your ISO image, just use the following structure, changing the command as necessary:

```
genisoimage ... -m *.html -m *.HTML -o backup.iso /home/huhn
```

The `-x` option lets you exclude whole directories; multiple arguments are supported:

```
genisoimage ... -x /tmp -x /var -o backup.iso /
```

When you use these parameters, make sure you avoid using wildcards when specifying the files you want to write to the image. For example, the command

```
genisoimage ... -m *.html -m *.HTML -o backup.iso *
```

tells the shell to resolve the final wildcard, adding all your files to the image.

If you want to exclude files with typical backup extensions, such as files ending in `~`, `#`, or `.bak`, just specify the `-no-bak` option.

Creating Bootable Media

A bootable medium can be used to start a computer. To create bootable, just add the *Isolinux* bootloader, which works hand in hand with `genisoimage` or `xorrisofs`:

```
genisoimage -J -R -o bootcd.iso -b isolinux/isolinux.bin -c isolinux/boot.cat -no-emul-boot -boot-load-size 4 -boot-info-table /folder/data
```

The additional *genisoimage* options in the preceding command are: `-b` to name the boot image and `-c` to specify the boot catalog. The `-no-emul-boot` parameter tells the program not to create an emulation when installing from this CD; instead, it writes the contents of the image file to disk. The `-boot-load-size 4` option specifies that the BIOS should provide four 512-byte sectors for the boot file. Finally, the `-boot-info-table` option stipulates that the layout information of the medium should be read at boot time. Note that this information must be stored in the *isolinux* directory below `/folder/data`.

Testing Images Before Burning

The 'mount' utility gives you a practical approach to testing ISO images before burning them to CD. To test your image, just mount it on your filesystem, specifying the `-o loop` option:

```
mount -o loop myfile.iso /mnt/tmp
```

The mountpoint must exist, and you do need root privileges for this command. After completing the test (Figure 1), you can unmount the ISO image again by entering `umount /mnt/tmp`.

Some distributions also include the `isoinfo` tool. You can access all sorts of information with it. To do so, specify the image file name after `-i`; `-d` outputs the report to the terminal.

If you want to find out which files are included in the ISO image, use `-l` instead of `-d`. Figure 1 shows the "content" of the Blu-ray image that was created with `genisoimage -udf`.

```
root@jessie:~/test# isoinfo -l -i blu-ray-test.iso

Directory listing of /
d----- 0 0 0          2048 Jun 30 2015 [ 267 02] .
d----- 0 0 0          2048 Jun 30 2015 [ 267 02] ..
----- 0 0 0 3976200192 Apr 27 2015 [ 268
00]DEBIAN_8_0_0_AMD64_DVD_1.ISO;1
----- 0 0 0 459570267 Apr 22 2015 [1941772
00]LINUXMINT_17_1_CINNAMON_64B.ISO;1
root@jessie:~/test#
```

INFO

[1] Isolinux: <http://syslinux.zytor.com/iso.php>

Device partitions and volumes

A hard disk on a modern computer is usually divided into partitions. A partition can contain exactly one filesystem (the data structure that stores files and directories). In Linux, swap space is usually implemented as a filesystem of its own, requiring its own partition. Additionally, many users create separate partitions for the `/boot`, `/var`, and `/home` directories. In theory, although Linux can be installed on one partition, likely you'll find multiple partitions.

Before you can install an operating system, you need to create partitions and format them with filesystems. Most Linux installers provide a GUI for creating and managing partitions during the installation process, but if your system is already installed, you can turn to several management utilities for configuring partitions. The Bash command line provides several utilities for creating and configuring partitions, including `fdisk`, `gdisk`, and `parted`.

These tools are all generally safe, but accidents such as power interruptions can happen, so be extremely careful. Before beginning, backup your data. Then boot from a Live disk so all your hard drive partitions are unmounted before you edit them. Most of all, check all your actions twice before beginning them.

MBR to GPT Switch

A disk is divided into sections called partitions. Until recently, information on the partition structure was stored in a small sector at the beginning of the disk known as the Master Boot Record (MBR).

The old MBR served the hard disk industry well, but the industry outgrew it. MBR-based disks could only have four primary partitions, and the size of a partition was limited to 2TB (once an impossibly large size but today sometimes severely limiting). The Unified Extensible Firmware Interface (UEFI) standard, which has rapidly gained acceptance in recent years, defines a new format for specifying partition information known as the GUID Partition Table (GPT). As older PCs are gradually replaced, GPT-based disks are gradually replacing MBR-based disks.

fdisk for MBR partitions

The *fdisk* utility lets you create and manage partitions on MBR-based disks. Fdisk has two basic modes: interactive and non-interactive. Non-interactive mode queries a partition and displays the information. By contrast, interactive mode is menu driven and lets you alter, as well as explore, partitions and partition tables.

Running `fdisk -l` prints a listing of the partition tables of all of the drives on the system. To use *fdisk*, you might need to preface it with *su* or *sudo* to attain root privileges, depending on your distribution. To view only a single drive's table, append the drive name to the command:==

```
fdisk -l /dev/sda
```

==Note that *fdisk* requires a drive as its device argument. Fdisk's output includes the total drive size and basic geometry, then lists the partitions on the drive and their start and end locations, size, and partition type (both by name and by ID number). Size is reported in blocks. By adding the `-u` flag, you can have *fdisk* report start and end locations in sectors instead of cylinders. Running

```
fdisk -s </some/device>
```

prints only the size of the device, but it works for both drives and partitions. To create or change partitions with *fdisk*, start it in interactive mode: Omit both the `-l` and `-s` flags, and specify a drive, such as

```
fdisk /dev/hda  
fdisk -u /dev/sdc
```

The program writes its output to the screen and provides a command prompt but does not provide paging (e.g., *less*), so you might need to scroll up to read lengthy output. ==Entering *m* at the prompt list the available fdisk commands. From the main menu, you can create new partitions (*n*), delete existing partitions (*d*), verify the partition table (*v*), and set several flags (the most notable being the bootable flag, toggled with *a*). To apply any changes, write a new partition table to the drive by entering *w*. At any time, you can quit without writing the partition table with *q*.==

==Creating a partition is a multistep process. Type *n* to begin, and *fdisk* will ask whether you want to create a primary partition (*p*) or an extended partition (*e*). Whichever you choose, *fdisk* will then ask you to select the partition number (be careful to choose an unused one if you have already created several), the location on the drive where you want the partition to start, and its size.==

==Fdisk will prompt you with the number of the first available cylinder on the drive. To leave an empty space between partitions, choose a higher number, which could help if you ever need to resize your partition.==

gdisk for GPT Partitions

The new UEFI standard replaces the old MBR with the new GPT format, which solves some of the problems associated with the MBR, supporting a much larger disk size and theoretically allowing up to 128 partitions on a disk.

Because GPT uses a different format for storing partition information, it requires a different utility. The most popular options for the Linux command line are the GPT fdisk utilities *gdisk*, *sgdisk*, and *fixparts*. The GPT fdisk toolset comes standard on several contemporary Linux systems; if you don't find it, install it through your distro's packaging system. Once you have installed the GPT toolset, you can use it to check and modify the disk. The *gdisk* command here returns what follows for a new disk:

```
gdisk /dev/sdh

Partition table scan:
MBR: not present
BSD: not present
APM: not present
GPT: not present
```

Several choices for managing the partition table appear in the text mode menu. For instance, choosing the *p* option prints the partition table, whereas *o* first outputs a security prompt and then creates a GPT table. The *n* option lets you create a new disk-sized data partition. The *w* option writes the data (saves the changes) from the *gdisk* session.

Gdisk can convert MBR-partitioned hard disks to GPT, which removes the need to back up and restore existing partition content. Choosing *r* sends you to the recovery and transformation menu, which offers options for converting your MBR disk to GPT. Gdisk can also convert from GPT to MBR. In some cases, this option will not work, so be sure to back up your data if you try it.

GNU Parted

Fdisk is one of the older Linux tools, and Gdisk is modeled after it. Increasingly, many users prefer GNU Parted.

==You can run *parted* in interactive or non-interactive mode. The non-interactive syntax is *parted </some/device>* . Running the command==

```
parted /dev/sda print
```

==prints the partition table found on the */dev/sda* disk. The information displayed is similar to *fdisk*'s.==

Providing the *-i* flag before the device and command arguments launches *parted* in interactive mode. Unlike *fdisk*, however, you can create and modify partitions in non-interactive mode as well.

One important difference between between *parted* and *fdisk* and *gdisk* is that *parted* commands take effect immediately; that is when you create a new partition table, the existing partition table (if any) on the target drive is overwritten. This leaves little margin for error when working with a drive that has existing partitions and can leave you hunting for recovery tools (fortunately, Parted can assist in that task as well). The command

```
parted </some/device> mklabel <type>
```

creates a new partition table on the specified device. You must specify the partition table type; for Linux, a widely used type is *msdos*, although Parted supports several others. To create a new partition, use:

```
parted </some/device> mkpart <partition_type> <filesystem_type> <start> <end>
```

where *partition_type* is *primary*, *extended*, or *logical*. For primary or logical partitions, you must also specify *filesystem_type*, the filesystem format that the partition will hold.

However, the *mkpart* command does not actually create the filesystem. To do so, use *mkpartfs* instead. Parted supports many common filesystems, including ext4, FAT32, NTFS, JFS, UFS, XFS, and Linux swap. The start and end parameters specify the location of the new partition on the disk; you do not have to use disk geometry such as sectors, but you can provide human-readable sizes expressed in megabytes.

To remove a partition, use *parted </some/device> rm* , where *is* the partition number.

Parted really improves on *fdisk* in its ability to move and resize partitions. For variety's sake, start Parted in interactive mode before exploring partition manipulation. You still need a drive device argument, such as

```
parted -i /dev/hdb
```

While in interactive mode, the device given as an argument is assumed; you do not need to include it in the commands you type. To switch to a different device within interactive mode, type

```
select <some/other/device>
```

at the Parted command prompt.

The *resize* command takes three arguments: the partition number, the new start location of the resized partition, and the new end location. To continue the above example, running

```
resize 1 0 1000
```

at the prompt would resize the partition at */dev/hdb1* to begin at the start of the drive and end at the 1000MB mark.

You can use Parted both to grow and shrink partitions. However, for ext2/3 filesystems, you cannot change the start location with a *resize* command, only the end. That restriction does not apply to the other filesystems that Parted supports.

Parted can move a partition to a new free location on a drive with the *move* command. The syntax is

```
move <partitionNumber> <start> <end>
```

although *is* optional. If omitted, the partition is moved to the new location in its original size. If an ending point defines a new size for the partition, Parted automatically resizes as well as moves it.

When shuffling and resizing partitions, you might need to duplicate a partition in a new location, perhaps to move a partition to a new device in an attempt to free up space. At the Parted prompt, use


```
cp <original/device> <originalPartition> <targetPartition>
```

where */original/device* is optional; if omitted, the current working device will be assumed. Thus, the command

```
cp /dev/sdb 5 1
```

copies the */dev/sdb5* partition to */dev/hdb1*.

If you accidentally delete a partition from the partition table or overwrite the partition table itself, entering

```
rescue <start> <end>
```

initiates a search for filesystem signatures on the disk. Parted searches a range of sectors around the start and end positions for signs of the filesystem, so you do not need to be exact. If it finds a potential filesystem in the appropriate location, Parted asks whether you want to create a new partition. For this rescue to work, the filesystem must be more or less intact; Parted can only recreate partition table entries - to fix filesystem corruption, you need other tools.

The LVM Alternative

A Logical Volume Manager (LVM) is an alternative to traditional partitioning that treats the space on one drive - or even multiple drives - as a single unit and divides it into logical volumes. GNU Parted also works with LVMs.

The use of LVMs may come at a price: Should a drive become corrupt, your entire system suddenly can be inaccessible. By contrast, if you use traditional partitions, especially for */home*, you might be able to recover data by booting from a Live device after the root or boot partition crashes. You should research the difference between LVMs and traditional partitioning carefully before deciding which to use.

Configuring filesystems with mkfs, df, du, and fsck

Linux supports a wide array of filesystem types, including many that originated on other operating systems. The most common choices for hard disks, however, remain the native ext3/4, followed by the high-performance XFS and Btrfs filesystems. For compatibility, knowing how to work with the VFAT filesystem is important, because it is the standard choice found pre-installed on many media, including USB thumb drives and flash disks. Additionally, several of the same utilities used to manage normal filesystems also apply to swap partitions, which the Linux kernel uses as virtual memory when RAM is scarce.

mkfs

The *mkfs* command creates a new filesystem on a specified block device, such as a partition on a hard disk. The basic usage is:

```
mkfs -t <filesystem_type> </the/device>
```

where `<filesystem_type>` is a Linux-supported filesystem type (e.g., `ext3` or `XFS`) and `</the/device>` is the location of the target disk partition (e.g., `/dev/sda1` or `/dev/sdc3`). Filesystem-specific options are added after `<filesystem_type>`.

Filesystem options

The `mkfs.` utility, where is a filesystem supported by the command (e.g., `ext3`, `ext4`, `XFS`, `Btrfs`, `VFAT`), supports options that tweak filesystem settings such as the size of blocks used, number and size of inodes, fragment size, amount of space reserved for root-privileged processes, amount of space reserved to grow the group block descriptor if the filesystem ever needs to be resized, and settings for stripe, stride, and other details required for using the filesystem in a RAID array.

All of these parameters have default settings, and unless you are sure you need to change them, you can safely create a filesystem with the default settings. Nevertheless, it is a good idea to familiarize yourself with the basics of filesystem parameters in general, in case you ever run into problems. The *block size* is the size of the chunks that the filesystem uses to store data - in a sense, it is the granularity of the pieces into which a file is split when stored on the disk.

Larger block sizes can improve disk throughput because the disk can read and write more data at a time before seeking to a new location; however, a large block size can waste space in the presence of many small files, because a full block is consumed for each fragment of a file, even if only a small portion of it is used. `Ext3/4` and `XFS` allow you to specify the block size (1024, 2048, 4096, etc.) by adding a `-b` flag; the syntax that follows the flag varies, so consult the manual pages for each option.

The `mkswap` command creates a swap area on a disk partition, just as `mkfs` creates a filesystem. The basic syntax is the same, `mkswap </the/swap/device>`, with the optional `-c` flag again allows you to check the partition for bad blocks before creating the swap area. Just as a new filesystem must be attached to Linux's root filesystem with `mount` before you can use it, a new swap partition must be attached with `swapon -L </the/swap/device>`.

The basic form of the command described in the previous example hands off creation of the filesystem to one of several specialized utilities, depending on the filesystem type you specify (e.g., `mkfs.ext3`, `mkfs.xfs`, `mkfs.vfat`). Because filesystems differ so much from each other, having specialized tools maintained by experts in the individual filesystems results in more stable code.

Most of these utilities implement the same options, although they vary according to the features implemented in the different filesystems. According to the `mkfs` man page, the general form of the command is now deprecated in favor of these type-specific `mkfs.*` utilities.

Despite the differences, a few key options are common to all `mkfs.*` utilities. Adding the `-c` flag checks the specified device for bad blocks, which is then skipped over during the filesystem creation step. Adding the `-v` or `-V` flags produces verbose or extremely verbose output, respectively.

Note: The simulated `mkfs` commands for `XFS` and `ext3` differ. (The `-N` and `-n` flags specify a simulation, which does not actually create a filesystem.) The `-f` and `-F` flags tell `mkfs` to force filesystem creation, even if it detects a filesystem already in place.

mkfs Examples

To format the first partition of the first drive on a system as `ext4`, you would run the command:

```
mkfs -t ext4 /dev/sda1
```

This command uses the default block size, inode parameters, and all other options, some of which are determined at run time when `mkfs` analyzes the geometry of the disk partition. Using

```
mkfs -t ext4 -b 4096 /dev/sda1
```

also creates an ext4 filesystem on `/dev/sda1`, but it forces the use of 4096-byte blocks. Running

```
mkfs -t ext4 -b 4096 -J device=/dev/sdb1 /dev/sda1
```

creates the same filesystem as the preceding command, but it creates the journal on a separate partition (`*/dev/sdb1`).

To create an XFS partition on `/dev/sda1`, enter the following `mkfs` command:

```
mkfs -t xfs /dev/sda1
```

To specify the use of 4096-byte blocks on this filesystem, use

```
mkfs -t xfs -b size=4096 /dev/sda1
```

which is a different syntax than that used for ext4. The following command, which uses the alternative (and now preferred) `mkfs.*`

```
mkfs.btrfs -L mylabel </dev/partition>
```

creates a Btrfs filesystem with a 16KiB default block size (where 1000KiB = 1024 KB). To create a partition with 4KiB block size, use:

```
mkfs.btrfs -L mylabel -1 4k </dev/partition>
```

The variations in syntax make it especially critical to refer to the man page for more on the use of `mkfs` with specific filesystem options.

Routine maintenance

Running out of space on a filesystem is one of the most common problems you are likely to encounter on a Linux system, and it is not just an inconvenience for storage reasons - the system's use of temporary files means that a full or nearly full root filesystem could interfere with normal operations.

To check filesystem usage, use `df`. When given no arguments, `df` returns a table summarizing usage of all of the mounted filesystems - in kilobytes and as a percentage of each filesystem's total size. To get a report for a particular filesystem, specify it as an argument, such as `df /dev/sda1`.

Also, you can pass a file name as an argument, and `df` will report on the filesystem that contains the specified file- which could be handy if you don't remember where a particular filesystem is mounted. Finally, a few options exist to make `df` more useful: `-i` reports inode usage instead of block usage of the filesystem(s); `-l` limits the report to local filesystems only; `--type= <`

`filesystem_type >` and `--exclude-type= <filesystem_type >` allow you to limit or exclude output to a particular filesystem type.

On discovering a nearly full filesystem, you can further explore space usage with `du`. Executing `du </some/directory>` returns a list of the disk space occupied by each subdirectory beneath `</some/directory>`, expressed in kilobytes. Adding the `-a` option tells `du` to report the space used by the files in addition to the directories.

Both commands are recursive. If you do not provide a directory as an argument to `du`, it reports on the current directory. The `-c` option produces a grand total in addition to individual usage statistics. Other helpful options are `-L`, which could help track down an errant large file, following all symbolic links; `-x`, which limits the scope of the search to the current filesystem only; and `--max-depth=N`, which allows you to limit the number of recursive subdirectories into which you descend. This option is very helpful when dealing with a large file library.

Several utilities exist to help you get better performance out of your filesystems. The `tune2fs` program lets you control many parameters of ext2, ext3, and ext4 filesystems. You can set the number of mounts between automatic filesystem integrity checks with `tune2fs -c N`, set the maximum time interval between checks with `tune2fs -i N[d|m|w]` (where `d`, `m`, and `w` are days, months and weeks, respectively), or add an ext3 or ext4 journal to a filesystem that does not have one with `tune2fs -j`. Additionally, you can adjust RAID parameters, journal settings, and reserved block behavior, as well as change parameters manually, such as the time last checked and number of mounts, which are usually reported automatically.

Other utilities are associated with specific filesystems. Btrfs has a separate utility for resizing filesystems (`btrfs filesystem resize`). The `*btrfs-convert` tool can migrate data from existing ext2/3/4 volumes to the Btrfs filesystem.

XFS also provides a defragmentation tool called `xfs_fsr` that can defragment a mounted XFS filesystem, and Btrfs supports defragmentation of metadata or entire filesystems. The

```
btrfs filesystem defragment -r -v /
```

command defragments the entire filesystem verbosely. No such utilities exist for ext3, but ext4 has `e4defrag`.

Troubleshooting

If you suspect trouble on a filesystem, you can run

```
fsck /a/<device>
```

to check and make repairs. If you run `fsck` with no target device specified, it will run checks sequentially on all of the filesystems in `/etc/fstab`.

The filesystem-specific error-checking programs - `e2fsck` for ext2, ext3 and ext4, `btrfsck` for Btrfs, and `fsck.vfat` for VFAT - support many of the same options, but again, the syntax may vary, so it is critical to read the man page for the filesystem checker before attempting any repairs.

When corrupted, VFAT filesystems suffer from bad clusters, bad directory pointers, and even bad file names. The `fsck.vfat` tool can find and correct many of these problems. Like the others, it can be called in non-interactive mode for use in scripts, and it can mark bad clusters automatically to prevent their reuse in the future. The `-V` flag tells `fsck.vfat` to run a second check after it has tried to correct any errors.

XFS has separate error-checking and repair utilities: *xfs_check* and *xfs_repair* (see the man pages for more on command-line options).

For ext2/3/4 problems, the *debugfs* tool lets you examine a filesystem and correct errors interactively. It can step through and work with commands similar to those of a typical Linux shell, such as *cd*, *open*, *close*, *pwd*, *mkdir*, and even *chroot*.

Media access with mount and fstab

Linux attaches media directly to the directory tree in a process known as mounting. Device files for devices of all kinds - network, removable media, hard disk partitions - can typically be found below the */dev* (device) directory.

Many modern Linux systems handle the mounting process automatically. Your hard drives, CDs, and USB sticks just appear in the desktop GUI, and you can navigate to the data without the need for explicit mounting commands; however, the classic Unix/Linux mount system is still useful for troubleshooting or when working with systems that don't offer udev support.

This article describes how to mount and manage storage resources using *mount*, *umount*, and the *fstab* file. Systemd environments offer an alternative option for mounting using *systemctl*.

The Systemd init environment used on most modern Linux systems lets you create a Systemd unit file then reference the file to mount the resource. The unit file should be named with the *.mount* extension to specify that the file contains mount information.

The unit file contains a *[Mount]* section with the following basic options:

- *What=*- path, partition name, or UUID for a device, partition, file, or other resource you wish to mount.
- *Where=*- absolute path of the mount point.
- *Type=*- (optional) the filesystem type

systemd.mount also supports several other unit file options. See the documentation online [1]. Once you create the unit file, you can mount and manage the resource using *systemctl* commands:

```
# systemctl daemon-reload
# systemctl start unit_file_name
```

See the article on Systemd (elsewherein this issue) for more unit files and the *systemctl* command.

INFO

[1] systemd.mount: <https://www.freedesktop.org/software/systemd/man/systemd.mount.html>

Name Game

Integrated Drive Electronics (IDE) device names (names of hard drives, CD-ROMs, or DVD drives) start with *sd* (the "s" refers to the Serial Advanced Technology Attachment (SATA) interface used for most modern mass storage); the letter that follows depends on the connector and the order. The first drive is *sda*, the second *sdb* and so on.

Linux handles USB mass storage devices, SD cards and so on as SCSI devices. Their device filenames also start with *sd*. CD and DVD drives tend to be listed as *sr*, and floppy disk drives are *fd*.

Besides the letters, many devices have numbers that reflect the logical structure of the storage media (e.g., the first primary partition on a SATA hard disk attached to the first controller is *sda1*, the second partition is *sda2*, etc.). Logical partition numbers start at 5. The second logical partition on *sdc* thus maps to the device file */dev/sdc6*.

The system enumerates SCSI CD/DVD drives in the same way (*sr0*, *sr1*, etc.), along with floppy drives (*fd0*, *fd1*, etc.).

Many distros use aliases such as */dev/cdrom* or */dev/dvd* that point to the names for CD/DVD drives. To support access to various devices and filesystems, you need to create a link between a device and a directory in the filesystem tree. The *mount* command associates a device with a directory.

Mounting

Mounting occurs at boot time or manually at a later stage. Hard disk partitions are usually mounted at boot time; USBs, CDs, DVDs, and other removable media used to be mounted manually and were often in the domain of the system administrator (root), unless the privilege was specifically given to users. Nowadays, most of these devices are autodetected and are mounted automatically or at a user prompt.

However, if you have trouble or need a little more control, the utility used to mount from the command line is *mount*. A number of optional parameters aside, you have to specify the device file and the mountpoint. If you call *mount* without supplying any parameters, the command tells you, among many other things, which media are currently mounted.

```
01 mount
02 /dev/sda1 on / type ext4 (rw,errors=remount-ro)
03 /dev/sda5 on /home type ext4 (rw)
04 /dev/sda6 on /mnt/scratch type ext3 (rw)
05 /dev/sr0 on /media/cdrom0 type iso9660 \
06 (ro,nosuid,nodev,utf8,user=huhn)
```

Additionally, *mount* tells you about the filesystems for the devices, and it lets you know what mount options are in place. The */dev/sda1* partition has been formatted with ext4 and mounted as the root partition (at */*); the CD drive contains ISO 9660 media (the default filesystem for data CDs) and has been mounted under */media/cdrom0*.

The listing also tells you if the hard disk partitions are readable and writable (*rw* for "read-write"). The information

```
errors=remount-ro
```

ensures the media will be remounted read-only; that is, the data will be readable but with no write access.

Mounting Removable Media

Data CDs/DVDs, floppy disks, and USB media are normally mounted automatically when you plug them in. If your system does not automount or you're working on the console, run *mount* manually. Linux assigns directories below */mnt* or */media* for removable media. In the command line, you need to type the device file name and the mountpoint.

When you mount a USB mass storage device, check the */var/log/messages* or */var/log/kern.log* logfiles to see if the device has been detected correctly and to discover the device file name. To mount the device detected, *sd*, in an existing directory, */media/usb*, type:

```
mount /dev/sdc1 /media/usb
```

Linux typically autodetects the filesystem type for media. If you get an error message, you can explicitly specify the filesystem by supplying a value for the *-t* parameter - for example,

```
mount -t vfat /dev/sdc1 /media/usb
```

for an older Windows filesystem on FAT-formatted media. Besides *vfat* (for the DOS/Windows filesystem), the supported values are *ext2* (extended filesystem version 2), *ext3* (extended filesystem version 3), *ext4* (extended filesystem version 4), *reiserfs* (Reiser filesystem), *iso9660* (ISO 9660), *ntfs* (NT filesystem), and so on.

Most systems define the device names and mountpoints for CDs/DVDs and floppies, so a command such as

```
mount /media/cdrom
```

might be all it takes to mount a CD.

Critical Mount Options

The *-o ro* option for *mount* makes a device "read-only." Its counterpart, as well as the default setting, is *-o rw* (for "read-write"). Combinations are also supported: To remove write access for media mounted with read-write access, supply two parameters when running the command; for example,

```
mount -o remount,ro /media/usb
```

tells *mount* to *remount* the media and at the same time disable write access (*ro*). To test an ISO by mounting a 1:1 copy of the image before burning, enter:

```
mount -o loop file_name.iso /mnt/tmp
```

which uses a loop device to access the image.

Tabular: /etc/fstab

Linux mounts some filesystems directly at boot time. The */etc/fstab* file has entries for the filesystems to mount.

The *fstab* file used to contain configuration information for the full set of hard disk partitions in addition to the various removable media. But, removable media are now managed by the *udev* subsystem, which allows regular users to mount and unmount them from the command line or desktop.

The first column is the device file, UUID, or label, and the second is the mountpoint. The other entries specify the filesystem for the media (the kernel normally autodetects this - *auto*), and various mount options.

Often you see entries such as *user* (the device can be mounted without root privileges), *nouser* (the opposite), *auto* (mounted at boot time), *noauto*, *exec* (executable), or *noexec*. If you want to modify the */etc/fstab* file, you must become root.

Out!

To unmount filesystems, use the command *umount*. Although Linux automatically dismounts mounted media at shutdown, you can also unmount devices manually, including removable CD-ROMs/DVDs, floppies, and USB devices:

```
umount /media/usb
umount /media/cdrom0
```

USB media and floppies must be unmounted before you remove them. CD and DVD drives block automatically and refuse to open the drive bay while a disk is mounted.

An additional safety mechanism is that *umount* will not unmount a filesystem while a process is accessing the files. A program might be using the data on the CD in the drive, or the data might be part of the working directory used by the shell or a file manager (i.e., */media/cdrom0*) or one of its subdirectories. To determine which process is blocking the device, run *lsof*, which displays open files and directories, as root against the device name of the drive, as in Listing 2.

```
# lsof /dev/sr0
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF  NODE NAME
gwenview  23884 paul   cwd   DIR   11,0    2048 4096 /run/media/paul/Ubuntu 15.04
amd64/pics
```

If *lsof* does not tell you what the command is, it will tell you the PID (process number). You can then use the *ps* tool and output a list of all processes in wide display mode, pipe the output to *grep*, and search the output for the process ID:

```
ps auxwww | grep 23884
  paul 23884  0.3 1.2 80453276544 ? S1   22:36  0.00 /usr/bin/gwenview
/run/media/paul/Ubuntu 15.04 amd64/ubuntu/pics/blue-lowerleft.png -caption
Gwenview -caption Gwenview --icon gwenview
```

In this case, it looks like *Gwenview* is the culprit. If you close the image viewer window showing the pictures on the CD content, you should be able to unmount the CD with:

```
umount /media/cdrom0
```

If this command doesn't help, you might have to be more assertive and use the *kill* command.

IDE Devices of Yore

Although they are very rare nowadays, you may have an old computer that still uses IDE-controlled mass storage devices. If this is the case, you need to know that IDE device names (names of hard disks, CD-ROMs, or DVD drives) start with *hd* (for *hard disk*); the letter that follows depends on the connector and the order. Each controller provides two connectors: The first is the master and the second is the slave. Device names are *hda* (master) and *hdb* (slave) for the devices attached to the first controller. Devices on the second controller are known as *hdc* (master) and *hdd* (slave).

The Language of Computers

Number Representations

- Analog - infinite number of intermediate states in between on and off
- Digital - discrete states
- Logical - true or false
- Mathematical - 0 or 1

bit - **binary digit**

Discrete states of binary information
on or off
yes or no
0 or 1
true or false

setting the bit - makes it a 1

resetting or clearing the bit - makes it a 0

8 bits is technically called an octet, but it is more commonly called a byte.

Number of Bits	Common Representation Terms
1	Bit/Digit/Flag
4	Nybble/Nibble
8	Byte/Octet/Character
16	Double Byte/Word
32	Double Word/Long Word
64	Very Long Word

Binary vs. decimal data measurements

Binary System

NAME	FACTOR	VALUE IN BYTES
kibibyte (KiB)	2^{10}	1,024
mebibyte (MiB)	2^{20}	1,048,576
gibibyte (GiB)	2^{30}	1,073,741,824
tebibyte (TiB)	2^{40}	1,099,511,627,776
pebibyte (PiB)	2^{50}	1,125,899,906,842,624
exbibyte (EiB)	2^{60}	1,152,921,504,606,846,976
zebibyte (ZiB)	2^{70}	1,180,591,620,717,411,303,424
yobibyte (YiB)	2^{80}	1,208,925,819,614,629,174,706,176

Decimal System

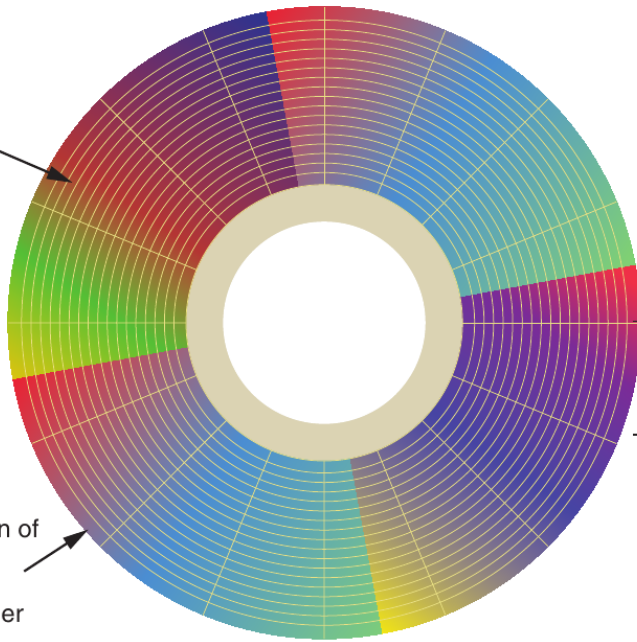
NAME	FACTOR	VALUE IN BYTES
kilobyte (KB)	10^3	1,000
megabyte (MB)	10^6	1,000,000
gigabyte (GB)	10^9	1,000,000,000
terabyte (TB)	10^{12}	1,000,000,000,000
petabyte	10^{15}	1,000,000,000,000,000
exabyte (EB)	10^{18}	1,000,000,000,000,000,000
zettabyte (ZB)	10^{21}	1,000,000,000,000,000,000,000
yottabyte (YB)	10^{24}	1,000,000,000,000,000,000,000,000

Understanding Disk Drives

You should be familiar with disk drives and how data is organized on a disk so that you can find data effectively. Disk drives are made up of one or more platters coated with magnetic material, and data is stored on platters in a particular way. The following disk drive components are illustrated in Figure 5-2:

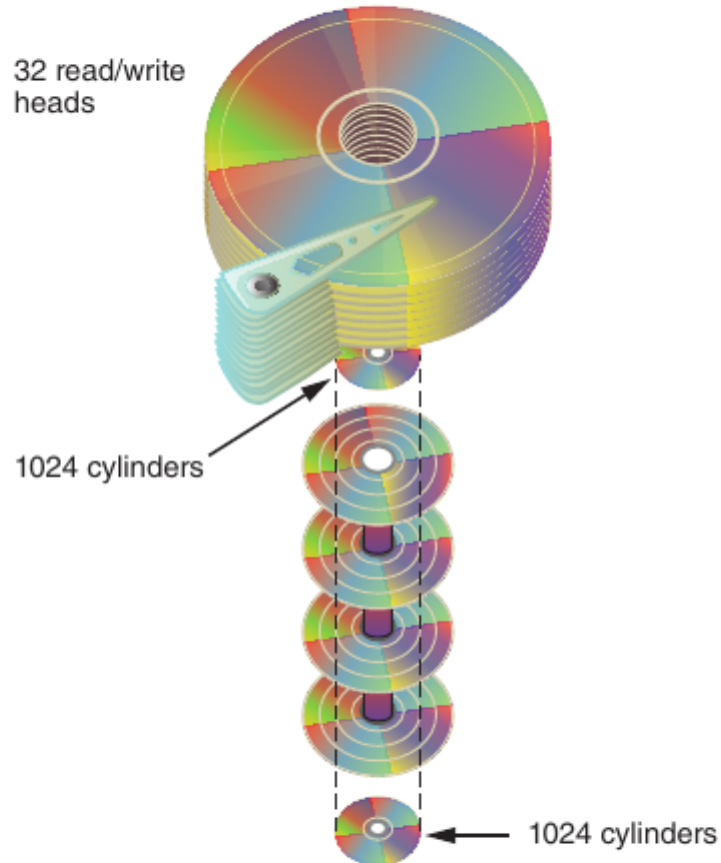
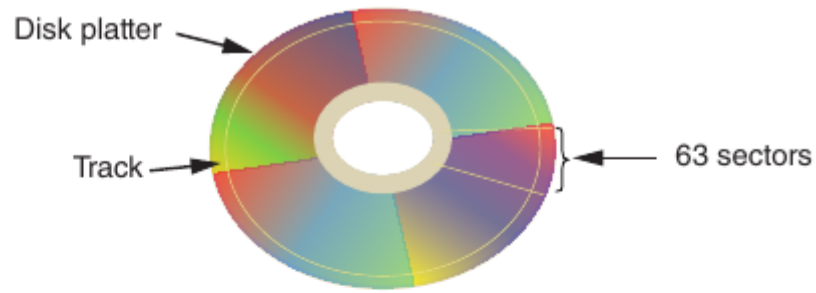
- Geometry—Geometry refers to a disk's logical structure of platters, tracks, and sectors.
- Head—The head is the device that reads and writes data to a drive. There are two heads per platter that read and write the top and bottom sides.
- Tracks—Tracks are concentric circles on a disk platter where data is located.
- Cylinders—A cylinder is a column of tracks on two or more disk platters. Typically, each platter has two surfaces: top and bottom.
- Sectors—A sector is a section on a track, usually made up of 512 bytes.

Each concentric circle is a track

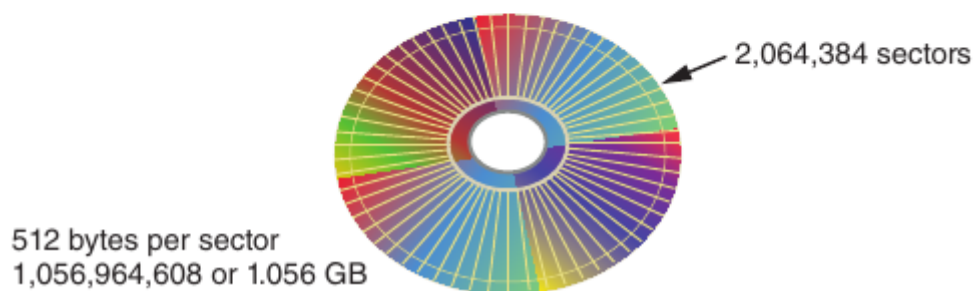


Each wedge-shaped area is a sector

Each combination of tracks forms a cylinder, which is stacked on another platter



$$1024 \text{ cylinders} \times 32 \text{ heads} \times 63 \text{ sectors} = 2,064,384 \text{ sectors}$$



Disk Geometry Equations

<https://blog.kalehmann.de/blog/2017/07/20/simple-boot-loader.html>

Parts of the Filesystem

In UNIX and Linux, everything is considered a file, including disk drives, monitors, tape drives, network interface cards, system memory, and directories. UNIX files are defined as objects, which means a file, like an object in an object-oriented programming language, has properties and methods (actions such as writing, deleting, and reading) that can be performed on it.

UNIX/Linux has four components defining the file system: boot block, superblock, inode block, and data block. A block is the smallest disk allocation unit in the UNIX/Linux file system and can be 512 bytes and up; block size depends on how the disk volume is initiated. The boot block contains the bootstrap code—instructions for startup. A UNIX/Linux computer has only one boot block, on the main hard disk.

The superblock contains vital information about the system and is considered part of the metadata. It specifies the disk geometry and available space and keeps track of all inodes (discussed in more detail in the following section). The superblock also manages the file system, including configuration information, such as block size for the drive, file system names, blocks reserved for inodes, and volume name. Multiple copies of the superblock are kept in different locations on the disk to prevent losing such important information.

Inode blocks contain the first data after the superblock. An inode is assigned to every file allocation unit. As files or directories are created or deleted, inodes are also created or deleted. The link between inodes associated with files and directories controls access to those files or directories.

The data block is where directories and files are stored on a disk drive. This location is linked directly to inodes. As in Microsoft file systems, the Linux file system on a PC has 512-byte sectors. A data block is equivalent to a cluster of disk sectors on a FAT or NTFS volume. Blocks range from 1024 to 4096 bytes each on a Linux volume.

Inodes

Inodes contain file and directory metadata and provide a mechanism for linking data stored in data blocks. When a file or directory is created on a Linux file system, an inode is assigned that contains the following information:

- The mode and type of the file or directory
- The number of links to a file or directory
- The UID and GID of the file's or directory's owner
- The number of bytes in the file or directory
- The file's or directory's last access time and last modified time
- The inode's last file status change time
- The block address for the file data
- The indirect, double-indirect, and triple-indirect block addresses for the file data
- Current usage status of the inode
- The number of actual blocks assigned to a file
- File generation number and version number
- The continuation inode's link

The only pieces of metadata not in an inode are the filename and path. Inodes contain modification, access, and creation (MAC) times, not filenames. An assigned inode has 15 pointers that link to data blocks and other pointers where files are stored. Pointers 1 through 12 link directly to data storage blocks in the disk's data block and contain block addresses indicating where data is stored on the disk. These pointers are direct pointers because each one is associated with one block of data storage.

As a file grows, the OS provides a few additional inode pointers. In a file's inode, the first 12 pointers are called direct pointers. The 13th pointer is called an indirect pointer. The 14th pointer is called a double-indirect pointer, and the 15th pointer is called a triple-indirect pointer.

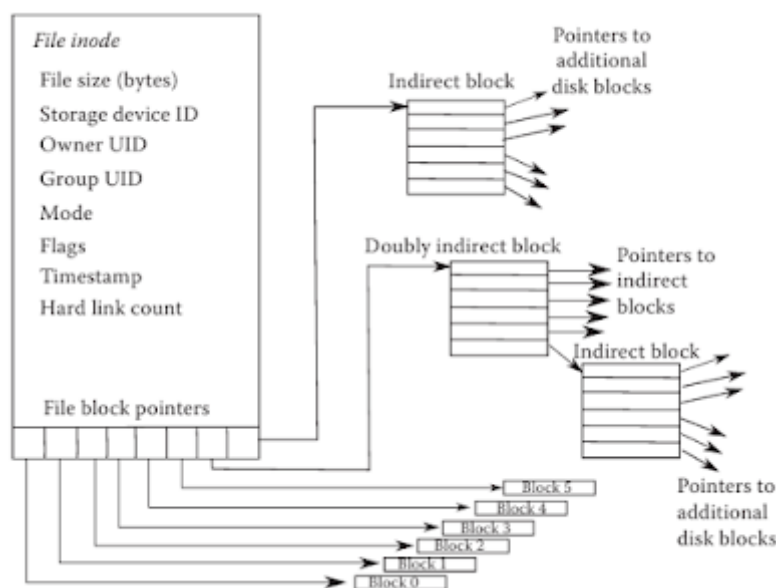
To expand storage allocation, the OS initiates the original inode's 13th pointer, which links to 128 pointer inodes. Each pointer links directly to 128 blocks located in the drive's data block. If all 12 pointers in the original inode are consumed with file data, the 13th pointer links to another 128 pointers. The first pointer in this indirect group of inodes points to the 13th block. The last block of these 128 inodes is block 130.

The `cp` creates a new file with a new inode number, whereas the `mv` command generates a copy of the original file, and it possesses the same inode as the original file, which is essentially a link. Hard links contain a filename and an inode number associated with that filename.

Nested Do Loop Inside For Loop (for moving files)

Note: The following command requires the use of an underscore character in the middle of every file that is to be moved.

```
for i in *_*;do mv $i <destination directory>;done
```



Source: <http://harrykar.blogspot.com/>

Let us take a look at how to access a Linux file through the inode. We will make a few assumptions.

First, our Linux inode will store 12 direct pointers, 1 indirect pointer, 1 doubly indirect pointer, and 1 triply indirect pointer. Blocks of pointers will store 12 pointers no matter whether they are indirect, doubly indirect, or triply indirect blocks. We will assume that our file consists of 500 blocks, numbered 0 to 499, each block storing 8 KB (the typical disk block stores between 1 KB and 8 KB depending on the file system utilized).

Our example file then stores $500 * 8 \text{ KB} = 4000 \text{ KB}$ or 4 MB. Here is the breakdown of how we access the various blocks.

- Blocks 0–11: direct pointers from the inode.
- Blocks 12–23: pointers from an indirect block pointed to by the inode's indirect pointer.
- For the rest of the file, access is more complicated.
 - We follow the inode's doubly indirect pointer to a doubly indirect block. This block contains 12 pointers to indirect blocks. Each indirect block contains 12 pointers to disk blocks.

- The doubly indirect block's first pointer points to an indirect block of 12 pointers, which point to blocks 24–35.
- The doubly indirect block's second pointer points to another indirect block of 12 pointers, which point to blocks 36–47.
- The doubly indirect block's last pointer points to an indirect block of 12 pointers, which point to blocks 156–167.
- We follow the inode's triply indirect pointer to a triply indirect block. This block contains 12 pointers to doubly indirect blocks, each of which contains 12 pointers to indirect blocks, each of which contain 12 pointers to disk blocks. From the triply indirect block, we can reach blocks 168 through 499 (with room to increase the file to block 1895).

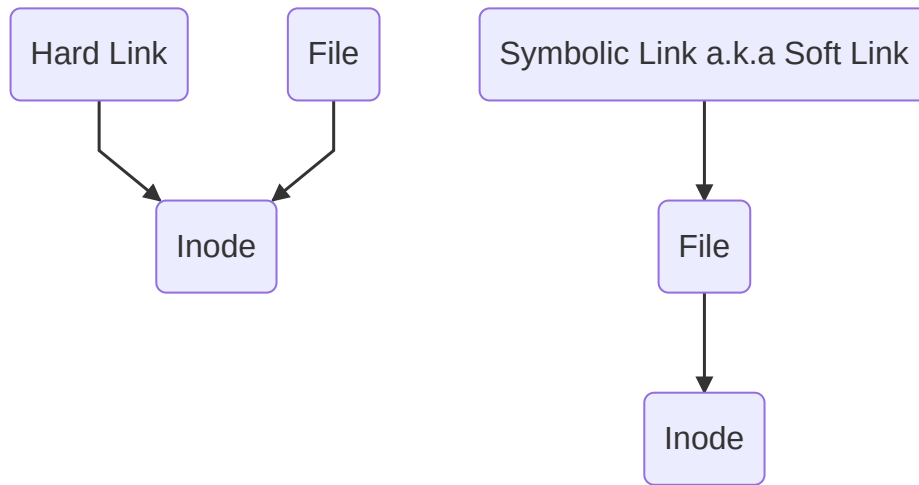
The Bad Block Inode

Linux keeps track of bad sectors in an inode called the **bad block inode**. The root inode is inode 2, and the bad block inode is inode 1. To find bad blocks on your Linux computer, you can use the `badblocks` command, although you must log in as root to do so. Linux includes two other commands that supply bad block information: `mke2fs` and `e2fsck`. The `badblocks` command can destroy valuable data, but the `mke2fs` and `e2fsck` commands include safeguards that prevent them from overwriting important information.

Inode-Related Commands

Command	Description
<code>stat file1.txt</code>	Display file's inode information
<code>stat --format=%i file1.txt</code>	Display file's inode number
<code>ls -i</code>	Display the inodes associated with each file in directory
<code>ls -i file1.txt</code>	Display the inode associated with a specific file
<code>df -i</code>	Display inode information associated with your mounted filesystems
<code>tune2fs -l /dev/sda2</code>	Display superblock information for a filesystem
<code>debugfs</code>	Manipulate filesystem metadata

Links



A hard link is a pointer that allows accessing the same file by different filenames (Rute-Users-Guide/Linux Dictionary V 0.16, www.tldp.org/LDP/Linux-Dictionary/html/index.html). The filenames refer to the same inode and physical location on a drive. Originally, hard links were used so that people with different logins could access the same physical file. If one person changed the file, the changes would be apparent when another user opened the file.

The main requirement is that all files pointing to the same inode have to be on the same physical drive, not on another volume. Inside each inode is a field called **link count** that specifies the number of hard links. The link count for directories is higher than for other file types. If two files have the same inode number, the link count is two. If one file is deleted, the link count drops by one. When To see the contents of a directory, you use the `ls -a` command. The first two items are `.` (called “dot”), which refers to the directory, and `..` (called “dot-dot”), which refers to the parent directory (see Figure 7-5). Both dot and dot-dot count as links, so most directories have at least two hard links. Every subdirectory has a dot-dot reference to the corresponding parent directory; therefore, each one adds to the parent directory’s link count.

Symbolic links (also known as “soft links” or “symlinks”) are simply pointers to other files and aren’t included in the link count. Unlike hard links, they can point to items on other drives or other parts of the network; they simply need an absolute path. Symbolic links have an inode of their own, which isn’t the same as the inode of the item they’re pointing to. Unlike hard links, they depend on the continued existence of the destination they’re pointing to, and they’re easier to identify on a running Linux system than hard links are. Unlike hard links, which point to their destination with an inode number, symbolic links identify their destination by name and path. If a name and path no longer exist, the symbolic link stops working. You create symbolic links with the `ln -s` command.

Commands for Hard Links & Soft Links

Hard Link	Soft Link
<code>ln ~/demo.txt ./demo-hard.txt</code>	<code>sudo ln -s ~/demo.txt ~/var/www/demo-soft.txt</code>

Filesystem Hierarchy Standard

Conforming to the Filesystem Hierarchy Standard (FHS), Version 3.0, published March 19, 2015 (<https://refspecs.linuxfoundation.org/fhs.shtml>).

Directory	Description
<code>/</code>	This is the root directory. This is where the whole tree starts.
<code>/bin</code>	This directory contains executable programs which are needed in single user mode and to bring the system up or repair it.
<code>/boot</code>	Contains static files for the boot loader. This directory holds only the files which are needed during the boot process. The map installer and configuration files should go to <code>/sbin</code> and <code>/etc</code> . The operating system kernel (<code>initrd</code> for example) must be located in either <code>/</code> or <code>/boot</code> .
<code>/dev</code>	Special or device files, which refer to physical devices.
<code>/etc</code>	Contains configuration files which are local to the machine. Some larger software packages, like X11, can have their own subdirectories below <code>/etc</code> . Site-wide configuration files may be placed here or in <code>/usr/etc</code> . Nevertheless, programs should always look for these files in <code>/etc</code> and you may have links for these files to <code>/usr/etc</code> .
<code>/etc/opt</code>	Host-specific configuration files for add-on applications installed in <code>/opt</code> .
<code>/etc/sgml</code>	This directory contains the configuration files for SGML (optional).
<code>/etc/skel</code>	When a new user account is created, files from this directory are usually copied into the user's home directory.
<code>/etc/X11</code>	Configuration files for the X11 window system (optional).
<code>/etc/xml</code>	This directory contains the configuration files for XML (optional).
<code>/home</code>	On machines with home directories for users, these are usually beneath this directory, directly or not. The structure of this directory depends on local administration decisions (optional).
<code>/lib</code>	This directory should hold those shared libraries that are necessary to boot the system and to run the commands in the root filesystem.
<code>/lib<qual></code>	These directories are variants of <code>/lib</code> on system which support more than one binary format requiring separate libraries (optional).

Directory	Description
<code>/lib/modules</code>	Loadable kernel modules (optional).
<code>/lost+found</code>	This directory contains items lost in the filesystem. These items are usually chunks of files mangled as a consequence of a faulty disk or a system crash.
<code>/media</code>	This directory contains mount points for removable media such as CD and DVD disks or USB sticks. On systems where more than one device exists for mounting a certain type of media, mount directories can be created by appending a digit to the name of those available above starting with '0', but the unqualified name must also exist.
<code>/media/floppy[1-9]</code>	Floppy drive (optional).
<code>/media/cdrom[1-9]</code>	CD-ROM drive (optional).
<code>/media/cdrecorder[1-9]</code>	CD writer (optional).
<code>/media/zip[1-9]</code>	Zip drive (optional).
<code>/media/usb[1-9]</code>	USB drive (optional).
<code>/mnt</code>	This directory is a mount point for a temporarily mounted filesystem. In some distributions, <code>/mnt</code> contains subdirectories intended to be used as mount points for several temporary filesystems.
<code>/opt</code>	This directory should contain add-on packages that contain static files.
<code>/proc</code>	This is a mount point for the proc filesystem, which provides information about running processes and the kernel. This pseudofilesystem is described in more detail in proc(5) .
<code>/root</code>	This directory is usually the home directory for the root user (optional).
<code>/run</code>	This directory contains information which describes the system since it was booted. Once this purpose was served by <code>/var/run</code> and programs may continue to use it.
<code>/sbin</code>	Like <code>/bin</code> , this directory holds commands needed to boot the system, but which are usually not executed by normal users.
<code>/srv</code>	This directory contains site-specific data that is served by this system.

Directory	Description
<code>/sys</code>	This is a mount point for the sysfs filesystem, which provides information about the kernel like <code>/proc</code> , but better structured, following the formalism of kobject infrastructure.
<code>/tmp</code>	This directory contains temporary files which may be deleted with no notice, such as by a regular job or at system boot up.
<code>/usr</code>	This directory is usually mounted from a separate partition. It should hold only shareable, read-only data, so that it can be mounted by various machines running Linux.
<code>/usr/X11R6</code>	The X-Window system, version 11 release 6 (present in FHS 2.3, removed in FHS 3.0).
<code>/usr/X11R6/bin</code>	Binaries which belong to the X-Window system; often, there is a symbolic link from the more traditional <code>/usr/bin/X11</code> to here.
<code>/usr/X11R6/lib</code>	Data files associated with the X-Window system.
<code>/usr/X11R6/lib/X11</code>	These contain miscellaneous files needed to run X; Often, there is a symbolic link from <code>/usr/lib/X11</code> to this directory.
<code>/usr/X11R6/include/X11</code>	Contains include files needed for compiling programs using the X11 window system. Often, there is a symbolic link from <code>/usr/include/X11</code> to this directory.
<code>/usr/bin</code>	This is the primary directory for executable programs. Most programs executed by normal users which are not needed for booting or for repairing the system and which are not installed locally should be placed in this directory.
<code>/usr/bin/mh</code>	Commands for the MH mail handling system (optional).
<code>/usr/bin/X11</code>	is the traditional place to look for X11 executables; on Linux, it usually is a symbolic link to <code>/usr/X11R6/bin</code> .
<code>/usr/dict</code>	Replaced by <code>/usr/share/dict</code> .
<code>/usr/doc</code>	Replaced by <code>/usr/share/doc</code> .

Directory	Description
<code>/usr/etc</code>	Site-wide configuration files to be shared between several machines may be stored in this directory. However, commands should always reference those files using the <code>/etc</code> directory. Links from files in <code>/etc</code> should point to the appropriate files in <code>/usr/etc</code> .
<code>/usr/games</code>	Binaries for games and educational programs (optional).
<code>/usr/include</code>	Include files for the C compiler.
<code>/usr/include/bsd</code>	BSD compatibility include files (optional).
<code>/usr/include/X11</code>	Include files for the C compiler and the X-Window system. This is usually a symbolic link to <code>/usr/X11R6/include/X11</code> .
<code>/usr/include/asm</code>	Include files which declare some assembler functions. This used to be a symbolic link to <code>/usr/src/linux/include/asm</code> .
<code>/usr/include/linux</code>	This contains information which may change from system release to system release and used to be a symbolic link to <code>/usr/src/linux/include/linux</code> to get at operating-system-specific information. (Note that one should have include files there that work correctly with the current libc and in user space. However, Linux kernel source is not designed to be used with user programs and does not know anything about the libc you are using. It is very likely that things will break if you let <code>/usr/include/asm</code> and <code>/usr/include/linux</code> point at a random kernel tree. Debian systems don't do this and use headers from a known good kernel version, provided in the libc*-dev package.)
<code>/usr/include/g++</code>	Include files to use with the GNU C++ compiler.
<code>/usr/lib</code>	Object libraries, including dynamic libraries, plus some executables which usually are not invoked directly. More complicated programs may have whole subdirectories there.
<code>/usr/libexec</code>	Directory contains binaries for internal use only and they are not meant to be executed directly by users shell or scripts.

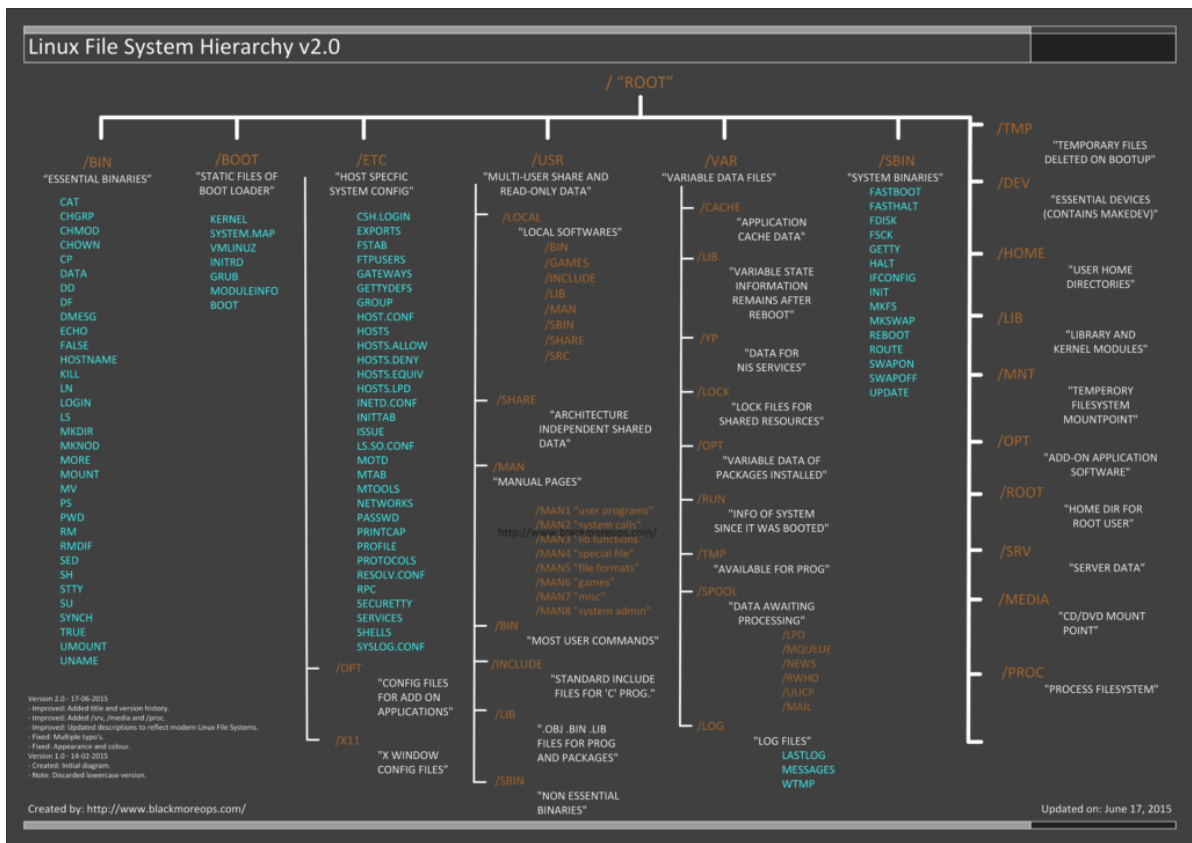
Directory	Description
<code>/usr/lib<qual></code>	These directories are variants of <code>/usr/lib</code> on system which support more than one binary format requiring separate libraries, except that the symbolic link <code>/usr/lib<qual>/X11</code> is not required (optional).
<code>usr/lib/X11</code>	The usual place for data files associated with X programs, and configuration files for the X system itself. On Linux, it usually is a symbolic link to <code>/usr/X11R6/lib/X11</code> .
<code>/usr/lib/gcc-lib</code>	contains executables and include files for the GNU C compiler, gcc (1).
<code>/usr/lib/groff</code>	Files for the GNU groff document formatting system.
<code>/usr/lib/uucp</code>	Files for uucp (1).
<code>/usr/local</code>	This is where programs which are local to the site typically go.
<code>/usr/local/bin</code>	Binaries for programs local to the site.
<code>/usr/local/doc</code>	Local documentation.
<code>/usr/local/etc</code>	Configuration files associated with locally installed programs.
<code>/usr/local/games</code>	Binaries for locally installed games.
<code>/usr/local/lib</code>	Files associated with locally installed programs.
<code>/usr/local/lib<qual></code>	These directories are variants of <code>/usr/local/lib</code> on system which support more than one binary format requiring separate libraries (optional).
<code>/usr/local/include</code>	Header files for the local C compiler.
<code>/usr/local/info</code>	Info pages associated with locally installed programs.
<code>/usr/local/man</code>	Man pages associated with locally installed programs.
<code>/usr/local/sbin</code>	Locally installed programs for system administration.
<code>/usr/local/share</code>	Local application data that can be shared among different architectures of the same OS.
<code>/usr/local/src</code>	Source code for locally installed software.
<code>/usr/man</code>	Replaced by <code>/usr/share/man</code> .

Directory	Description
<code>/usr/sbin</code>	This directory contains program binaries for system administration which are not essential for the boot process, for mounting <code>/usr</code> , or for system repair.
<code>/usr/share</code>	This directory contains subdirectories with specific application data, that can be shared among different architectures of the same OS. Often one finds stuff here that used to live in <code>/usr/doc</code> or <code>/usr/lib</code> or <code>/usr/man</code> .
<code>/usr/share/color</code>	Contains color management information, like International Color Consortium (ICC) Color profiles (optional).
<code>/usr/share/dict</code>	Contains the word lists used by spell checkers (optional).
<code>/usr/share/dict/words</code>	List of English words (optional).
<code>/usr/share/doc</code>	Documentation about installed programs (optional).
<code>/usr/share/games</code>	Static data files for games in <code>/usr/games</code> (optional).
<code>/usr/share/info</code>	Info pages go here (optional).
<code>/usr/share/locale</code>	Locale information goes here (optional).
<code>/usr/share/man</code>	Manual pages go here in subdirectories according to the man page sections.
<code>/usr/share/man/<locale>/man[1-9]</code>	These directories contain manual pages for the specific locale in source code form. Systems which use a unique language and code set for all manual pages may omit the substring.
<code>/usr/share/misc</code>	Miscellaneous data that can be shared among different architectures of the same OS.
<code>/usr/share/nls</code>	The message catalogs for native language support go here (optional).
<code>/usr/share/ppd</code>	Postscript Printer Definition (PPD) files (optional).
<code>/usr/share/sgml</code>	Files for SGML (optional).
<code>/usr/share/sgml/docbook</code>	DocBook DTD (optional).
<code>/usr/share/sgml/tei</code>	TEI DTD (optional).
<code>/usr/share/sgml/html</code>	HTML DTD (optional).
<code>/usr/share/sgml/mathml</code>	MathML DTD (optional).

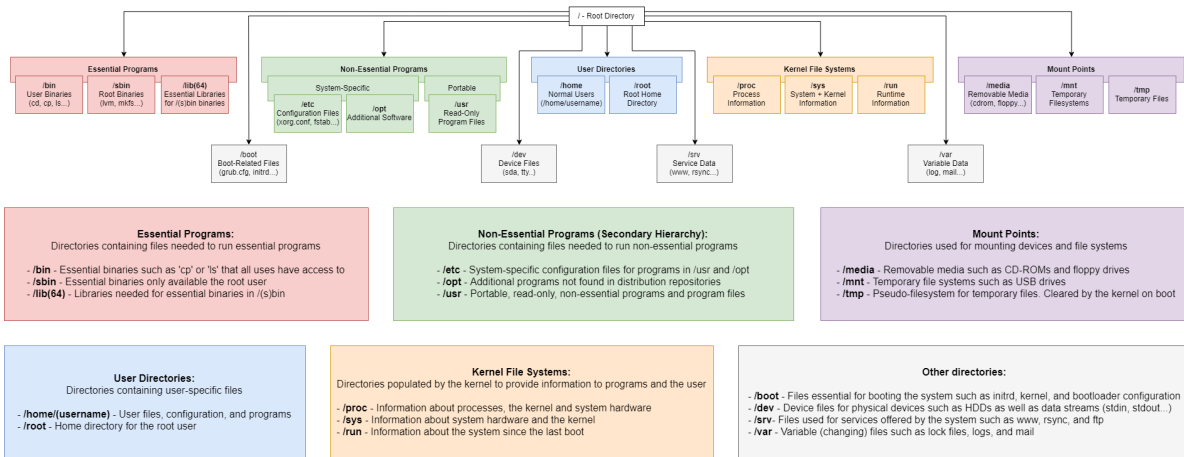
Directory	Description
<code>/usr/share/terminfo</code>	The database for terminfo (optional).
<code>/usr/share/tmac</code>	Troff macros that are not distributed with groff (optional).
<code>/usr/share/xml</code>	Files for XML (optional).
<code>/usr/share/xml/docbook</code>	DocBook DTD (optional).
<code>/usr/share/xml/xhtml</code>	XHTML DTD (optional).
<code>/usr/share/xml/mathml</code>	MathML DTD (optional).
<code>/usr/share/zoneinfo</code>	Files for timezone information (optional).
<code>/usr/src</code>	Source files for different parts of the system, included with some packages for reference purposes. Don't work here with your own projects, as files below <code>/usr</code> should be read-only except when installing software (optional).
<code>/usr/src/linux</code>	This was the traditional place for the kernel source. Some distributions put here the source for the default kernel they ship. You should probably use another directory when building your own kernel.
<code>/usr/tmp</code>	Obsolete. This should be a link to <code>/var/tmp</code> . This link is present only for compatibility reasons and shouldn't be used.
<code>/var</code>	This directory contains files which may change in size, such as spool and log files.
<code>/var/account</code>	Process accounting logs (optional).
<code>/var/adm</code>	This directory is superseded by <code>/var/log</code> and should be a symbolic link to <code>/var/log</code> .
<code>/var/backups</code>	Reserved for historical reasons.
<code>/var/cache</code>	Data cached for programs.
<code>/var/cache/fonts</code>	Locally-generated fonts (optional).
<code>/var/cache/man</code>	Locally-formatted man pages (optional).
<code>/var/cache/www</code>	WWW proxy or cache data (optional).
<code>/var/cache/<package></code>	Package specific cache data (optional).
<code>/var/catman/cat[1-9]</code> or <code>/var/cache/man/cat[1-9]</code>	These directories contain preformatted manual pages according to their man page section. (The use of preformatted manual pages is deprecated.)
<code>/var/crash</code>	System crash dumps (optional).

Directory	Description
<code>/var/cron</code>	Reserved for historical reasons.
<code>/var/games</code>	Variable game data (optional).
<code>/var/lib</code>	Variable state information for programs.
<code>/var/lib/color</code>	Variable files containing color management information (optional).
<code>/var/lib/hwclock</code>	State directory for hwclock (optional).
<code>/var/lib/misc</code>	Miscellaneous state data.
<code>/var/lib/xdm</code>	X display manager variable data (optional).
<code>/var/lib/<editor></code>	Editor backup files and state (optional).
<code>/var/lib/<name></code>	These directories must be used for all distribution packaging support.
<code>/var/lib/<package></code>	State data for packages and subsystems (optional).
<code>/var/lib/<pkgtool></code>	Packaging support files (optional).
<code>/var/local</code>	Variable data for <code>/usr/local</code> .
<code>/var/lock</code>	Lock files are placed in this directory. The naming convention for device lock files is LCK.. where is the device's name in the filesystem. The format used is that of HDU UUCP lock files, that is, lock files contain a PID as a 10-byte ASCII decimal number, followed by a newline character.
<code>/var/log</code>	Miscellaneous log files.
<code>/var/opt</code>	Variable data for <code>/opt</code> .
<code>/var/mail</code>	Users' mailboxes. Replaces <code>/var/spool/mail</code> .
<code>/var/msg</code>	Reserved for historical reasons.
<code>/var/preserve</code>	Reserved for historical reasons.
<code>/var/run</code>	Run-time variable files, like files holding process identifiers (PIDs) and logged user information (utmp). Files in this directory are usually cleared when the system boots.
<code>/var/spool</code>	Spooled (or queued) files for various programs.
<code>/var/spool/at</code>	Spooled jobs for at (1).
<code>/var/spool/cron</code>	Spooled jobs for cron (8).
<code>/var/spool/lpd</code>	Spooled files for printing (optional).
<code>/var/spool/lpd/printer</code>	Spools for a specific printer (optional).

Directory	Description
<code>/var/spool/mail</code>	Replaced by <code>/var/mail</code> .
<code>/var/spool/queue</code>	Queued outgoing mail (optional).
<code>/var/spool/news</code>	Spool directory for news (optional).
<code>/var/spool/rwhod</code>	Spooled files for rwhod (8) (optional).
<code>/var/spool/smmail</code>	Spooled files for the smmail (1) mail delivery program.
<code>/var/spool/uucp</code>	Spooled files for uucp (1) (optional).
<code>/var/tmp</code>	Like <code>/tmp</code> , this directory holds temporary files stored for an unspecified duration.
<code>/var/yp</code>	Database files for NIS, formerly known as the Sun Yellow Pages (YP).



The Filesystem Hierarchy Standard (FHS)



Created by Max Hölzl and licensed under the Creative Commons **CC-BY 4.0** license. Last edit: 2018-05-20

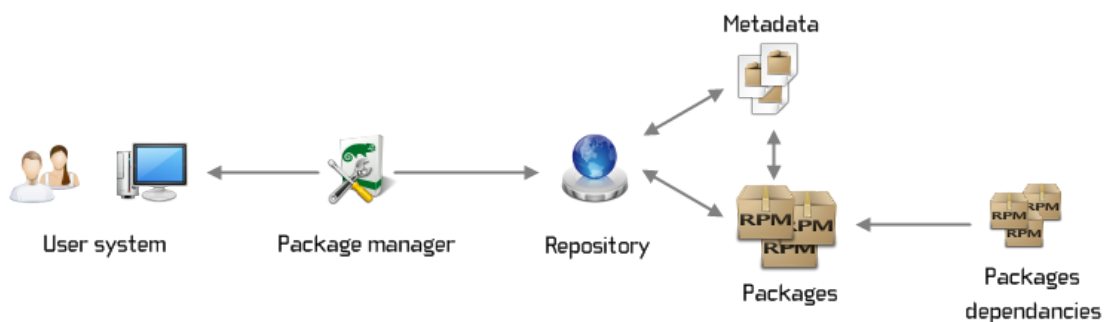
Package Management

Package managers are used to automate the process of installing, upgrading, configuring, and removing programs. Package managers are also used for installing and managing modules for languages such as Python, Ruby, etc.

A package is simply an archive that contains binaries of software, configuration files, and information about dependencies.

How Package Management Systems Work

If a certain package requires a certain resource, such as a shared library, or another package, it is said to have a dependency. All modern package management systems provide some method of dependency resolution to ensure that when a package is installed, all of its dependencies are installed as well.



Source: https://en.opensuse.org/Package_management

Package Tools

To perform the task of package management effectively, be aware of these two types of available utilities:

- Low-Level tools
- High-Level tools

Distribution	Low Level Tool	High Level Tool
Debian and derivatives	<code>dpkg</code>	<code>apt-get</code> / <code>aptitude</code>
CentOS	<code>rpm</code>	<code>yum</code>
openSUSE	<code>rpm</code>	<code>zypper</code>

Usage of Low-Level Tools

- Installing a packages from a compiled (*.deb or *.rpm) file
- Upgrading a package from a compiled file
- Listing installed packages
- Finding out which package installed a file

Usage of High-Level Tools

- Searching for a package
- Installing a package from a repository
- Removing a package
- Displaying information about a package

Table 1: apt and apt-get Compared

Function	apt	apt-get
Display package information	<code>dpkg list</code>	<code>list</code>
Find available packages	<code>apt-cache search</code>	<code>search</code>
Display package information	<code>apt-cache show</code> , <code>dpkg-query --list</code>	<code>show</code>
Install software packages	<code>install</code>	<code>install</code>
Remove software packages	<code>remove</code>	<code>remove</code>
Remove no longer needed packages	<code>autoremove</code>	<code>autoremove</code>
Update sources list (updates the repositories that are in use)	<code>update</code>	<code>update</code>
Upgrade installed packages	<code>dist-upgrade</code>	<code>upgrade</code>
Upgrade installed packages, removing packages to avoid conflicts	<code>dist-upgrade</code>	<code>full-upgrade</code>
Edit repository list	Open <code>/etc/apt/sources.list</code> in text editor	<code>edit sources</code>

Restrict repository list at `/etc/apt/sources.list` to the continental United States:

```
sudo sed -i -e 's/http:\\\\us.archive/mirror:\\\\mirrors/' -e 's
/\\/ubuntu\\/\\/mirrors.txt/' /etc/apt/sources.list
```

When you install software in Linux, dependencies (the necessary libraries and utilities) are added automatically. Years ago, the Debian package system was the first to include this feature, and a whole ecosystem of utilities has grown up around it. Soon after, most other distributions added their own dependency-resolving features. Today, software installations rarely fail because a dependency wasn't installed. The dreaded "dependency hell" is mostly a thing of the past, unless you try to install from poorly maintained third-party sources or mix packages from different repositories.

Linux includes far too many different package managers to cover them all in a single article. Presented here are only the most widely used ones, from Debian and Fedora, along with the so-called universal package managers.

Debian and Debian Derivatives

Aptitude stuff

```
aptitude search mysql      # Look for something
dpkg -S `which tsclient`   # What package does it belong to?
dpkg -L aria2c             # What does this package provide?
dpkg -i *.deb              # Install a deb file
dpkg -s nodejs             # Show info

dpkg --get-selections      # list installed packages
```

Apt archives path

```
/var/cache/apt/archives
```

List services

```
service --status-all
```

Debian and derivatives like Linux Mint and Ubuntu manage packages with `dpkg`. Many graphical interfaces are available, but for complete control, the only practical solution is the `dpkg` front end, Advanced Package Tool (APT). APT can be configured from `/etc/apt/preferences`, although most people prefer to configure it through sub-commands and options.

The `apt` command combines the basic options of `apt-get` and popular utilities such as `apt-cache` into a single command and adds a progress bar. If you ever need the old tools, or simply prefer them, they are still installed by default.

Unlike most commands, `apt` and `apt-get` consist of three parts separated by spaces: the basic `apt-get` command, a sub-command, and the packages involved in the operation. In practice, `apt` rarely needs options, having folded a few common ones into sub-commands.

A sub-command must always be present for both `apt` and `apt-get`, but for some maintenance tasks, you do not need a specific list of packages. To include multiple packages, either list the packages separated by a space or use regular expressions, such as the asterisk, although doing so can make troubleshooting more difficult and sometimes lead to unforeseen results.

If the packages to install are related, look for a metapackage, which is a dummy package meant to simplify the installation of large applications that are split into more than one package. For instance, to run the Gnome desktop environment in Debian, `gnome` saves you the effort of installing dozens of packages separately. To see whether a metapackage exists for your purposes, search online in your distro's repositories; if all else fails, guess its name, and see whether you are successful.

Depending on whether you are using `apt-get` or `apt`, the basic command for adding or upgrading a software package is either

```
apt-get install options <packagename>
```

or

```
apt install <packagename>
```

Deleting a package uses the same structure, except the sub-command is `remove`.

Both `apt` and `apt-get` usually start with a complete summary of what will happen if you go through with the installation, including the dependencies that will be installed, the packages that will be upgraded and removed, and the amount of disk space that will be required.

How to Use Git

Download for Linux and Unix

It is easiest to install Git on Linux using the preferred package manager of your Linux distribution. If you prefer to build from source, you can find tarballs [on kernel.org](https://kernel.org). The latest version is [2.34.1](https://kernel.org).

Debian/Ubuntu

For the latest stable version for your release of Debian/Ubuntu

```
# apt-get install git
```

For Ubuntu, this Personal Package Archive (PPA) provides the latest stable upstream Git version

```
# add-apt-repository ppa:git-core/ppa`# apt update; apt install git
```

Fedora

```
# yum install git (up to Fedora 21)
```

```
# dnf install git (Fedora 22 and later)
```

Gentoo

```
# emerge --ask --verbose dev-vcs/git
```

Arch Linux

```
# pacman -S git
```

openSUSE

```
# zypper install git
```

Mageia

```
# urpmi git
```

Nix/NixOS

```
# nix-env -i git
```

FreeBSD

```
# pkg install git
```

Solaris 9/10/11 ([OpenCSW](#))

```
# pkgutil -i git
```

Solaris 11 Express

```
# pkg install developer/versioning/git
```

OpenBSD

```
# pkg_add git
```

Alpine

```
$ apk add git
```

Red Hat Enterprise Linux, Oracle Linux, CentOS, Scientific Linux, et al.

RHEL and derivatives typically ship older versions of git. You can [download a tarball](#) and build from source, or use a 3rd-party repository such as [the IUS Community Project](#) to obtain a more recent version of git.

Slitaz

```
$ tazpkg get-install git
```

Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

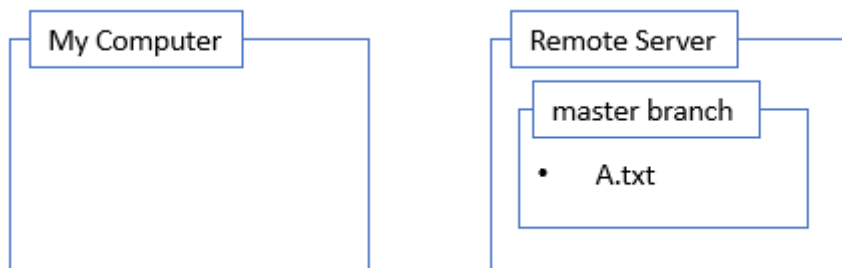
You can also always browse the current contents of the git repository using the [web interface](#).

The follow information on how to use Git is from a blog post at [The Bare Minimum You Need To Know To Work With Git](#). My thanks to Michael J Swart for this.

Initial Setup

One-time tasks include [downloading git](#) and signing up for [github](#) or [bitbucket](#). My team uses github, but yours might use gitlab, bitbucket or something else.

Here's my typical workflow. Say I want to work on some files in a project on a remote server:

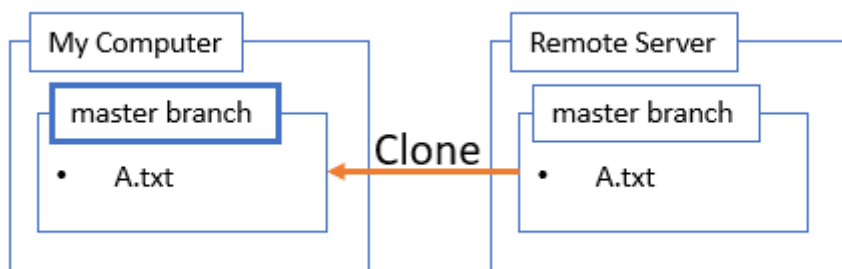


Clone a Repository

My first step is to find the repository for the project. Assuming I'm not starting a project from scratch, I find and copy the location of the repository from a site like github or bitbucket. So the clone command looks like this:

```
git clone https://github.com/SomeProject/SomeRepo.git
```

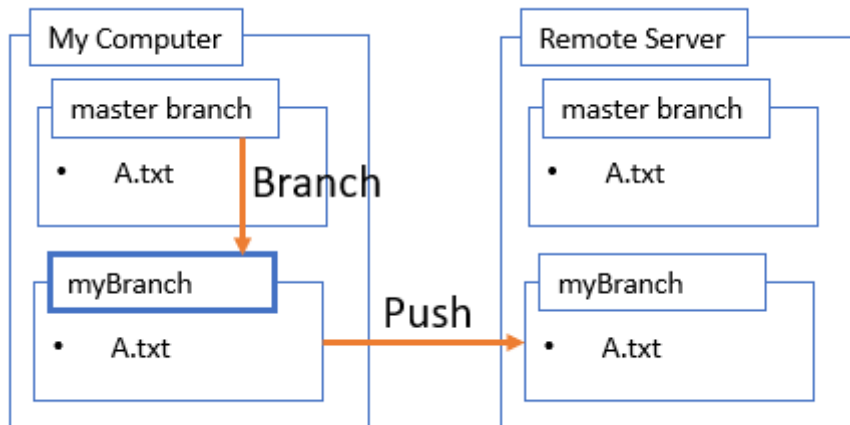
This downloads all the files so I have my own copy to work with.



Create a Branch

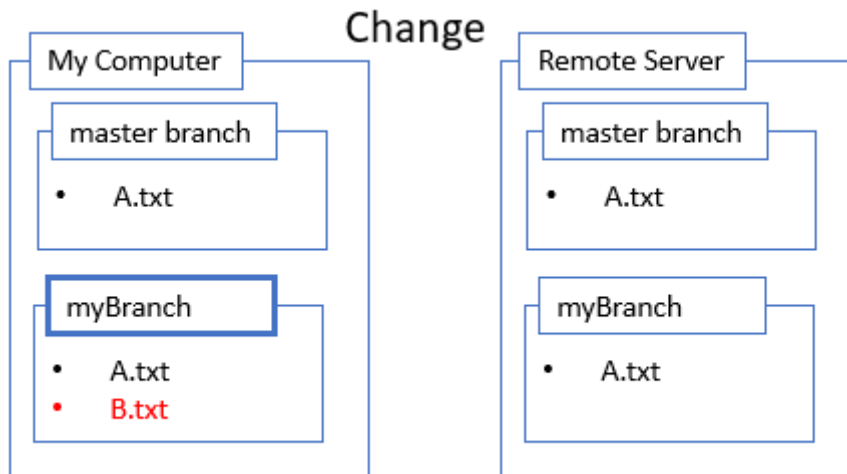
Next I create a branch. Branches are “alternate timelines” for a repository. The real timeline or branch is called master. One branch can be checked out at a time, so after I create a branch, I check out that branch. In the diagram, I've indicated the checkout branch in bold. I like to immediately push that branch back to the remote server. I can always refer to the remote server as “origin”. All this is done with these commands:

```
git branch myBranch
git checkout myBranch
git push -u origin myBranch
```



Change Stuff

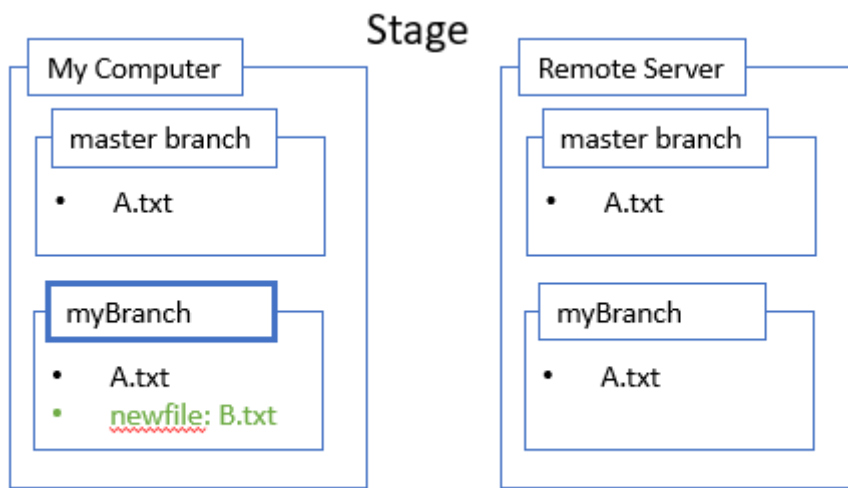
Now it's time to make changes. This has nothing to do with git but it's part of my workflow. In my example here I'm adding a file `B.txt`.



Stage Changes

These changes aren't part of the branch yet though! If I want them to be part of the branch, I have to commit my changes. That's done in two parts. The first part is to specify the changes I want to commit. That's called staging and it's done with `git add`. I almost always want to commit everything, so the command becomes:

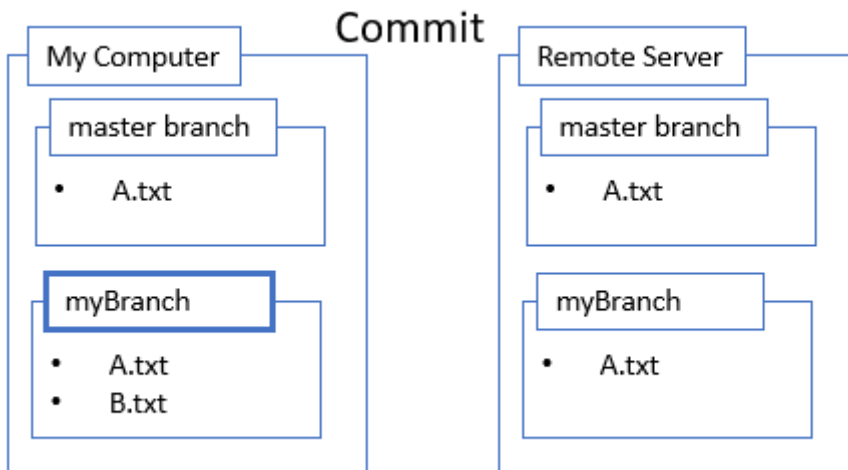
```
git add *
```



Commit Changes

The second part is to actually commit the files to the branch with a commit message:

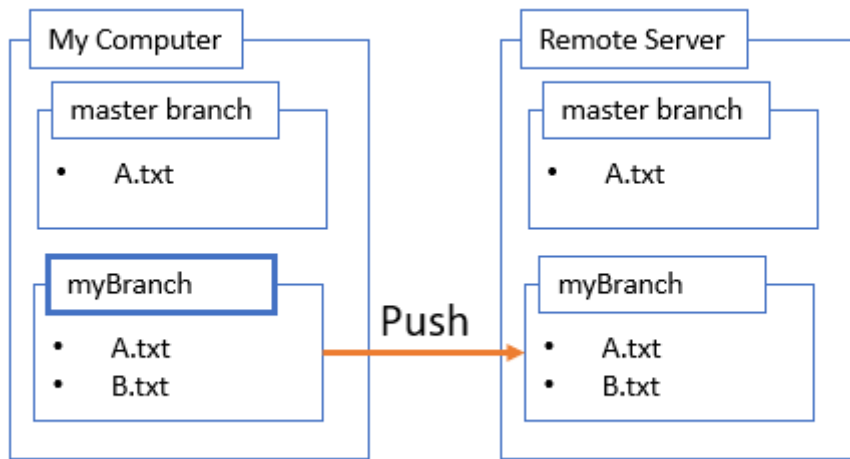
```
git commit -m "my commit message"
```



Push Changes

I'm happy with the changes I made to my branch so I want to share them with the rest of the world starting with the remote server.

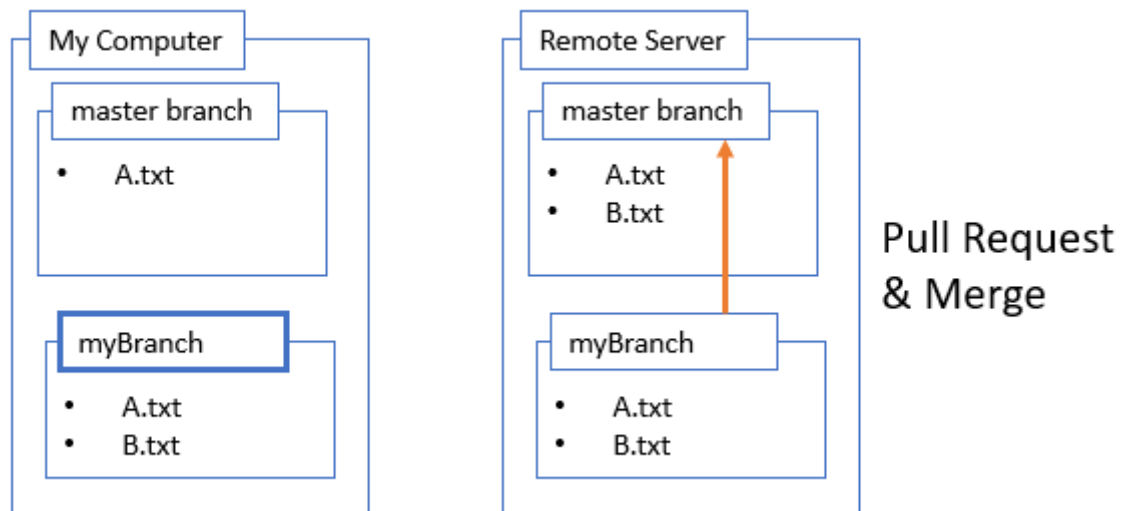
```
git push origin myBranch
```



Create a Pull Request and Merge to master

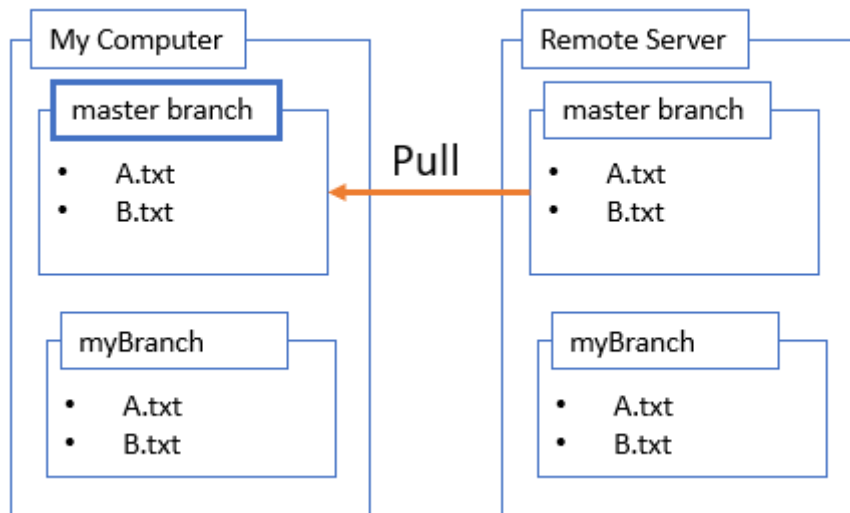
In fact I'm so happy with these changes, I want to include them in master, the real timeline. But not so fast! This is where collaboration and teamwork become important. I create a pull request and then if I get the approval of my teammates, I can merge.

It sounds like a chore, but luckily I don't have to memorize any git commands for this step because of sites like github or bitbucket. They have a really nice web site and UI where teams can discuss changes before approving them. Once the changes are approved and merged, master now has the changes.



Once it's merged. Just to complete the circle, I can pull the results of the merge back to my own computer with a pull

```
git pull git checkout master
```



Other Use Cases

Where Am I?

To find out where I am in my workflow, I like to use:

```
git status
```

This one command can tell me what branch I'm on. Whether there are changes that can be pushed or pulled. What files have changed and what changes are staged.

Merge Conflicts

With small frequent changes, merge conflicts become rare. But they still happen. Merge conflicts are a pain in the neck and to this day I usually end up googling "resolving git merge conflicts".

Can't this Be Easier?

There are so many programs and utilities available whose only purpose is to make this stuff easier. But they don't. They make some steps easy, and some steps impossible. Whenever I really screw things up, I delete everything and start from scratch at the cloning step. I find I have to do that more often when I use a tool that was supposed to make my life easier.

One Exception

The only exception to this rule is [Visual Studio Code](#). It's a real treat to use. I love it.

Maybe you like the command line. Maybe you have a favorite "git-helper" application. No matter how you use git, in every case, you still have to understand the workflow you're using and that's what I've tried to describe here.

Where To Next

If you want to really get good at this stuff. I recently learned of a great online resource (thanks Cressa!) at <https://learngitbranching.js.org/>. It's a great interactive site that teaches more about branching. You will very quickly learn more than the bare minimum required. I recommend it.

Git

Is the world's most advanced distributed version control system

Supplementary Instructions

Many people know that Linus created the open source Linux in 1991. Since then, the Linux system has continued to develop and has become the largest server system software. Although Linus created Linux, the growth of Linux depends on the participation of enthusiastic volunteers all over the world. With so many people writing code for Linux all over the world, how is the Linux code managed? The fact is that before 2002, volunteers around the world sent source code files to Linus through diff, and Linus himself merged the code manually!

You might be thinking, why doesn't Linus put Linux code in a version control system? Isn't there a free version control system like CVS and SVN? Because Linus is staunchly against CVS and SVN, these centralized version control systems are slow and must be networked to use. There are some commercial version control systems. Although they are easier to use than CVS and SVN, they are paid and do not match the open source spirit of Linux.

However, by 2002, the Linux system had been developed for ten years, and the size of the code base made it difficult for Linus to continue to manage it manually. Brothers in the community also expressed strong dissatisfaction with this method, so Linus chose a commercial version control system, BitKeeper. The owner of BitKeeper, BitMover, out of humanitarian spirit, authorizes the Linux community to use this version control system free of charge.

(The following paragraph is from Wikipedia.) Git development began in April 2005, after many developers of the [Linux kernel](#) gave up access to [BitKeeper](#), a proprietary source-control management (SCM) system that they had been using to maintain the project since 2002.^{[13][14]} The copyright holder of BitKeeper, [Larry McVoy](#), had withdrawn free use of the product after claiming that [Andrew Tridgell](#) had created [SourcePuller](#) by [reverse engineering](#) the BitKeeper protocols.^[15] BitMover was angry and wanted to take back the free use rights of the Linux community. The same incident also spurred the creation of another version-control system, [Mercurial](#).

Git quickly became the most popular distributed version control system, especially in 2008, when the GitHub website went live, it provided free Git storage for open source projects, and countless open source projects began to migrate to GitHub, including jQuery, PHP, Ruby, and more.

History is such a coincidence. If BitMover hadn't threatened the Linux community, we might not have free and super easy-to-use Git now.

List of common Git commands

```
git [--version] [--help] [-C <path>] [-c name=value]
  [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
  [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
  [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
  <command> [<args>]
```

Options

add	# adds file contents to the index
bisect	# introduces erroneous changes via binary lookup
branch	# list, create or delete branches

```
checkout      # Check out a branch or path to the working tree

clone         # clones the repository into a new directory

commit       # records the changes to the repository

diff         # shows changes between commits, commits and working tree etc.

fetch        # downloads objects and references from another repository

grep         # print lines matching the pattern

init         # creates an empty Git repository or reinitializes an existing
one

log          # shows the commit log

merge        # join two or more development histories

mv           # move or rename a file, directory or symbolic link

pull         # fetch and merge from another repository or local branch

push         # updates remote references and related objects

rebase       # forwards port local commits to updated upstream headers

reset        # resets the current HEAD to the specified state

rm           # removes files from working tree and index

show         # displays various types of objects

status       # Show working tree status

tag          # Create, list, delete or verify tag objects signed with GPG
```

example

init

```
git init      # initialization
```

status

```
git status    # Get status
```

add

```
git add file  # . or * means add all
```

rm

```
git rm --cached <added_file_to_undo> # undo the git add operation before commit
```

reset

```
git reset head # seems to be more convenient than the above git rm --cached
```

commit

```
git commit -m "message" #Pay attention to garbled characters here
```

remote

```
git remote add origin git@github.com:JSLite/test.git # Add source
```

push

```
git push -u origin master # push colleagues to set the default tracking branch
```

```
git push origin master
```

```
git push -f origin master # Force push files, abbreviated -f (full write --force)
```

clone

```
git clone git://github.com/JSLite/JSLite.js.git
```

```
git clone git://github.com/JSLite/JSLite.js.git mypro #Clone to custom folder
```

```
git clone [user@]example.com:path/to/repo.git/ # There is another way of writing the SSH protocol.
```

git clone supports multiple protocols, in addition to HTTP(s), it also supports SSH, Git, local file protocol, etc. Here are some examples.

```
git clone <URL of repository> <local directory name>
```

```
$ git clone http[s]://example.com/path/to/repo.git/
```

```
$ git clone ssh://example.com/path/to/repo.git/
```

```
$ git clone git://example.com/path/to/repo.git/
```

```
$ git clone /opt/git/project.git
```

```
$ git clone file:///opt/git/project.git
```

```
$ git clone ftp[s]://example.com/path/to/repo.git/
```

```
$ git clone rsync://example.com/path/to/repo.git/
```

configure

The first is to configure the account information `ssh -T git@github.com test`.

Modify personal information in a project

```
git help config # Get help information and view the parameters for
modifying personal information
```

```
git config --global user.name "little brother tune" # Modify the global name
```

```
git config --global user.email "wowohoo@qq.com" # Modify the global mailbox
```

```
git config --list # View configuration information
```

Configure word wrapping

The automatic conversion pit is too large, and submitting to git automatically converts newlines to lf

```
git config --global core.autocrlf input
```

Common usage scenarios

Create SSH keys

This key is used to communicate with github, generated in the local terminal and uploaded to github.

```
ssh-keygen -t rsa -C 'wowohoo@qq.com' # Generate key
ssh-keygen -t rsa -C "wowohoo@qq.com" -f ~/.ssh/ww_rsa # Specify the name of the
generated directory file
ssh -T git@github.com # test whether it is
successful
```

Multi-account ssh configuration

1. Generate a key with the specified name.

```
ssh-keygen -t rsa -C "email address" -f ~/.ssh/jslite_rsa # 2 files,
jslite_rsa and jslite_rsa.pub, are generated.
```

2. Copy the key to the hosting platform.

```
vim ~/.ssh/jslite_rsa.pub
```

Open the public key file jslite_rsa.pub and copy the content to the code hosting platform

3. Modify the config file

```
vim ~/.ssh/config #Modify the config file, if no config is created
```

```
Host-jslite.github.com
```

```
HostName github.com
```


User git

IdentityFile ~/.ssh/jslite_rsa

Hostwork.github.com

HostName github.com

Port server open-ssh port (default: 22, this line is generally not written by default)

PreferredAuthentications Configure which authorization authentication to use when logging in

publickey|password publickey|keyboard-interactive, etc.

User git

IdentityFile ~/.ssh/work_rsa

Host is an alias here, you can name it whatever you want

HostName is generally a website such as: [git@ss.github.com](https://github.com):username/repo.git Fill in github.com

User usually fills in git

The public key file address used by IdentityFile

\4. Test

ssh -T [git@jslite.github.com](https://github.com) # @ followed by the defined Host

ssh -T work.github.com # pass alias test

ssh -i ~/public key file address Host alias # Such as ssh -i ~/.ssh/work_rsa work.github.com

\5. Use

Original spelling

git clone [git@github.com](https://github.com):/learngit.git

present writing

git clone [git@jslite.github.com](https://github.com):/learngit.git

git clone [git@work.github.com](https://github.com):<work's username>/learngit.git

\5. Pay attention

If you change the name of id_rsa, you need to add the ssh key to the SSH agent, like:

ssh-add ~/.ssh/jslite_rsa

ssh-add -l # View all keys

ssh-add -D # delete all keys

ssh-add -d ~/.ssh/jslite_rsa # delete the specified key

Log in to a remote server without a password

```
$ ssh-keygen -t rsa -P "" -f ~/.ssh/aliyunserver.key
```

```
$ ssh-copy-id -i ~/.ssh/aliyunserver.key.pub root@192.168.182.112 # Here you need to enter the password once
```

Edit ~/.ssh/config

```
Host aliyun1
```

```
HostName 192.168.182.112
```

```
User root
```

```
PreferredAuthentications publickey
```

```
IdentityFile ~/.ssh/aliyunserver.key
```

After the above configuration is complete, you can log in through the command without entering the IP address and password `ssh aliyun1`

Submit code under https protocol without password

```
git clone https://github.com/username/rep.git
```

Cloning through the above method may require a password, the solution: enter the currently cloned project `vi rep/.git/config` to edit the config, and modify it in the following way, you can submit the code without entering the password.

```
[core]
```

```
repositoryformatversion=0
```

```
filemode=true
```

```
bare = false
```

```
logallrefupdates=true
```

```
ignorecase = true
```

```
precomposeunicode=true
```

```
[remote "origin"]
```

```
- url=https://github.com/username/rep.git
```

```
+ url = https://username:password@github.com/username/rep.git
```

```
fetch = +refs/heads/:refs/remotes/origin/
```

```
[branch "master"]
```

```
remote = origin
```

```
merge = refs/heads/master
```

File push to 3 git repositories

\1. Add 3 remote library addresses

```
git remote add origin https://github.com/JSLite/JSLite.git
```

```
git remote set-url --add origin https://gitlab.com/wang/JSLite.js.git
```

```
git remote set-url --add origin https://oschina.net/wang/JSLite.js.git
```

\2. Delete one of the set-url addresses

```
usage: git remote set-url [--push] []
```

```
or: git remote set-url --add
```

```
or: git remote set-url --delete
```

```
git remote set-url --delete origin https://oschina.net/wang/JSLite.js.git
```

\3. Push code

```
git push origin master
```

```
git push -f origin master # Force push
```

\4. Pull the code

Only one url address in origin can be pulled, this fetch-url

Defaults to the first address you add to origin

```
git pull origin master
```

```
git pull --all # Get all remote content including tags
```

```
git pull origin next:master # Retrieve the next branch of the origin host and merge it with the local master branch
```

```
git pull origin next # The remote branch is merged with the current branch
```

The above command is equivalent to the following two commands

```
git fetch origin
```

```
git merge origin/next
```

If the remote host deletes a branch, by default, git pull will not delete the corresponding local branch when pulling the remote branch. This is to prevent git pull from unknowingly deleting the local branch due to someone else operating the remote host.

However, you can change this behavior by adding the parameter -p to delete the remote deleted branch locally.

```
$ git pull -p
```

Equivalent to the following command

```
$ git fetch --prune origin
```

```
$ git fetch -p
```

\5. Change the pull

Just change the order of the three urls in the config file, and the fetch-url will directly correspond to the first url connection.

Modify the remote warehouse address

```
git remote remove origin # delete the remote path
```

```
git remote add origin git@jslite.github.com:JSLite/JSLite.git # Add remote path
```

Undo a remote record

```
git reset --hard HEAD~1 # undo a record
```

```
git push -f origin HEAD:master # Sync to remote repository
```

Discard local file modifications

```
git reset --hard FETCH_HEAD # FETCH_HEAD represents the commit point formed after the last successful git pull. then git pull
```

Error with git reset --hard FETCH_HEAD

```
git pull
```

You are not currently on a branch, so I cannot use any

'branch..merge' in your configuration file.

Please specify which remote branch you want to use on the command

line and try again (e.g. 'git pull ').

See git-pull(1) FOR details.

Solution:

```
git checkout -b temp # New + switch to temp branch
```

```
git checkout master
```

Easiest to give up on local modifications

```
# If there are any modifications and additions to the staging area
```

```
git reset --hard
```

```
# Restore all modifications, will not delete newly added files
```

```
git checkout .
```

```
# The following command will delete the newly added files
```

```
git clean -xdf
```

By storing the stash in the staging area, the local modification is discarded in the method of deleting the staging area.

```
git stash && git stash drop
```

```
rollback to a commit
```

```
git revert HEAD~1 # undoing a record will pop up commit edit
```

```
git push # commit rollback
```

```
fall back to a certain version
```

```
git reset --hard
```

```
# eg git reset --hard a3hd73r
```

--hard stands for discarding the modification of the workspace, so that the workspace is exactly the same as the version code, corresponding to it,

The --soft parameter represents the modification of the reserved workspace.

```
remove a commit
```

The essence is to create a new commit that is completely opposite to the original, offsetting the effect of the original commit

```
git revert
```

```
Create an empty branch
```

The new branch (gh-pages) in this way has no commit record

```
git checkout --orphan gh-pages
```

Delete the original content of the newly created gh-pages branch. If it is not deleted, the commit will be used as the first commit of the current branch

```
git rm -rf .
```

Check the status It is possible that the above command did not delete the files that have not been submitted

```
git state
```

```
Merge multiple commits
```

This command merges the last 4 commits into 1, HEAD represents the current version.

Will enter the VIM interface, you can modify the submission information.

```
git rebase -i HEAD~4
```

You can see that it is divided into two parts, the uncommented part above is to fill in the instructions to be executed,

And the part of the comment below is the prompt description of the command. The command part consists of the preceding command name, commit hash and commit message

Currently we only need to know the two commands pick and squash.

--> pick means to execute this commit

--> squash means that this commit will be merged into the previous commit

We change the command in front of the commit that we need to keep to squash or s, then enter :wq to save and exit

This is the edit interface where we will see the commit message

Among them, the non-comment part is the two commit messages, all you have to do is to modify these two into new commit messages.

#

Enter wq to save and launch, enter git log again to view the commit history information, you will find that these two commits have been merged.

Force push changes to the frontend

```
git push -f origin master
```

Modify remote Commit records

```
git commit --amend
```

amend can only modify the last commit record that has not been submitted online

```
git rebase -i HEAD~3
```

Indicates that you want to modify the penultimate state of the current version

Change the first word pick of the record line to be changed to edit

```
pick 96dc3f9 doc: Update quick-start.md
```

```
pick f1cce8a test(Transition): Add transition test (#47)
```

```
pick 6293516 feat(Divider): Add Divider component.
```

Rebase eeb03a4..6293516 onto eeb03a4 (3 commands)

#

Commands:

p, pick = use commit

r, reword = use commit, but edit the commit message

e, edit = use commit, but stop for amending

s, squash = use commit, but meld into previous commit

f, fixup = like "squash", but discard this commit's log message

x, exec = run command (the rest of the line) using shell

d, drop = remove commit

Save and exit, the following prompt will pop up

You can amend the commit now, with

#

git commit --amend

#

Once you are satisfied with your changes, run

#

git rebase --continue

Enter the edit page to change the commit through this command, save and exit

git commit --amend

Save and exit to confirm the modification, continue to execute rebase,

git rebase --continue

If multiple records are modified, execute the above two commands repeatedly until all modifications are completed

Finally, make sure that others have not submitted for push, it is best not to add -f to force push

git push -f origin master

Add ignore file

echo node_modules/ >> .gitignore

Close an issue with commit

This function can be played on Github, but not on very old versions on Gitlab, so how to close an issue with a commit? When confirm merge, you can use the following command to close related issues:

fixes #xxx, fixed #xxx, fix #xxx, closes #xxx, close #xxx, closed #xxx,

Synchronize the fork's upstream repository

Github tutorial synchronization fork tutorial, synchronize a branch (fork) on Github

Set to add multiple remote warehouse addresses.

Before synchronizing, you need to create a remote point to point to the upstream repository (repo). If you have forked an original repository, you can do as follows.

\$ git remote -v

List the current remotes (list the current remote warehouse)

origin <https://github.com/user/repo.git> (fetch)

```
# origin https://github.com/user/repo.git (push)
$ git remote add upstream https://github.com/otheruser/repo.git
# Set a new remote
$ git remote -v
# Verify new remote
# origin https://github.com/user/repo.git (fetch)
# origin https://github.com/user/repo.git (push)
# upstream https://github.com/otheruser/repo.git (fetch)
# upstream https://github.com/otheruser/repo.git (push)

Synchronously update warehouse contents
```

Syncing the upstream repository to your repository requires two steps: first you need to pull from the remote, then you need to merge the branch you want to your local copy branch. Pull branches and their respective commits from the upstream repository. master will be stored in the local branch upstream/master

```
git fetch upstream
# remote: Counting objects: 75, done.
# remote: Compressing objects: 100% (53/53), done.
# remote: Total 62 (delta 27), reused 44 (delta 9)
# Unpacking objects: 100% (62/62), done.
# From https://github.com/ORIGINAL\_OWNER/ORIGINAL\_REPOSITORY
# * [new branch] master -> upstream/master

Check out your fork's local master branch
```

```
git checkout master
# Switched to branch 'master'
```

Merge changes from upstream/master into the local master branch. This keeps your pre-fork's master branch in sync with the upstream repository without losing your local changes.

```
git merge upstream/master
# Updating a422352..5fdff0f
# Fast-forward
# README | 9 -----
# README.md | 7 ++++++
# 2 files changed, 7 insertions(+), 9 deletions(-)
```



```
# delete mode 100644 README
```

```
# create mode 100644 README.md
```

Batch modify names and email addresses in historical commits

\1. Clone the repository

Pay attention to the parameters, this is not an ordinary clone, the cloned warehouse cannot participate in the development

```
git clone --bare https://github.com/user/repo.git
```

```
cd repo.git
```

\2. Run the code from the command line

OLD_EMAIL old email address

CORRECT_NAME corrected name

CORRECT_EMAIL Corrected email

Copy the following code to the command line to execute

```
git filter-branch -f --env-filter '
```

```
OLD_EMAIL="wowohoo@qq.com"
```

```
CORRECT_NAME="Brother Tune"
```

```
CORRECT_EMAIL="Corrected Email@qq.com"
```

```
if [ "$GIT_COMMITTER_EMAIL" = "$OLD_EMAIL" ]
```

```
then
```

```
    export GIT_COMMITTER_NAME="$CORRECT_NAME"
```

```
    export GIT_COMMITTER_EMAIL="$CORRECT_EMAIL"
```

```
fi
```

```
if [ "$GIT_AUTHOR_EMAIL" = "$OLD_EMAIL" ]
```

```
then
```

```
    export GIT_AUTHOR_NAME="$CORRECT_NAME"
```

```
    export GIT_AUTHOR_EMAIL="$CORRECT_EMAIL"
```

```
fi
```

```
' --tag-name-filter cat -- --branches --tags
```

Implementation process

Rewrite 160d4df2689ff6df3820563bfd13b5f1fb9ba832 (479/508) (16 seconds passed, remaining 0 predicted)

Ref 'refs/heads/dev' was rewritten

Ref 'refs/heads/master' was rewritten

\3. Sync to remote warehouse

Sync to push remote git repository

```
git push --force --tags origin 'refs/heads/*'
```

I also encountered the following error, lab protects the master branch by default, and does not allow forced coverage. The Protected branches under the Project->Setting->Repository menu can remove the protection of the master. After the modification, it is recommended to add the protection of the master back. After all, pushing hard is not a good thing.

remote: GitLab: You are not allowed to force push code to a protected branch on this project.

When the above push does not go up, first git pull to ensure the latest code

```
git pull --allow-unrelated-histories
```

or specify branch

```
git pull origin master --allow-unrelated-histories
```

View a file's history

```
git log --pretty=oneline filename # list all change history of the file
```

```
git show c178bf49 # The modification record of a certain change
```

```
git log -p c178bf49 # The modification record of a certain change
```

```
git blame filename # Show that each line of the file was last modified in that version.
```

```
git whatchanged filename # Display the commit information of each version of a file: commit date, commit person, version number, commit notes (no modification details)
```

Build your own git command

```
git config --global alias.st status
```

```
git config --global alias.br branch
```

```
git config --global alias.co checkout
```

```
git config --global alias.ci commit
```

When you enter the git command after configuration, you don't need to enter a long paragraph. For example, if we want to check the status, just:

```
git st
```

Chinese garbled solution

```
git config --global core.quotePath false
```

New warehouse

init

```
git init #initialization
```

status

```
git status #Get status
```

add

```
git add file # . or * means add all
```

```
git rm --cached <added_file_to_undo> # undo the git add operation before commit
```

```
git reset head # seems to be more convenient than the above git rm --cached
```

commit

```
git commit -m "message" #Pay attention to garbled characters here
```

remote

```
git remote add origin git@github.com:JSLite/test.git #Add source
```

push

```
git push -u origin master # push colleagues to set the default tracking branch
```

```
git push origin master
```

```
git push -f origin master # Force push files, abbreviated -f (full write --force)
```

clone

```
git clone git://github.com/JSLite/JSLite.js.git
```

```
git clone git://github.com/JSLite/JSLite.js.git mypro #Clone to custom folder
```

```
git clone [user@]example.com:path/to/repo.git/ #There is another way of writing the SSH protocol.
```

git clone supports multiple protocols, in addition to HTTP(s), it also supports SSH, Git, local file protocol, etc. Here are some examples. git clone

```
$ git clone http[s]://example.com/path/to/repo.git/
```

```
$ git clone ssh://example.com/path/to/repo.git/
```

```
$ git clone git://example.com/path/to/repo.git/
$ git clone /opt/git/project.git
$ git clone file:///opt/git/project.git
$ git clone ftp[s]://example.com/path/to/repo.git/
$ git clone rsync://example.com/path/to/repo.git/

local

help

git help config # Get help information

add

git add * # track new files
git add -u [path] # Add a tracked file [under the specified path]

rm

rm *&git rm * # remove files
git rm -f * # remove files
git rm --cached * # untracking
git mv file_from file_to # rename tracked files
git log # View commit records

commit

git commit #Submit update
git commit -m 'message' #Commit description
git commit -a #Skip the use of the staging area and temporarily store all tracked files and submit them together
git commit --amend #Modify the last commit
git commit log #View all commits, including commits without push
git commit -m "#133" #Associate the issue with the # symbol and the issue number anywhere
git commit -m "fix #133" commit closes the issue
git commit -m 'Summary description'$'\n\n"1.Detailed description'$'\n"2.Detailed description'
#Commit brief description and detailed description

reset

git reset HEAD * # Cancel already staged files
git reset --mixed HEAD * # same as above
git reset --soft HEAD * # Reset to the specified state without modifying the index area and working tree
git reset --hard HEAD * # Reset to the specified state, which will modify the index area and working tree
git reset -- files * # reset index area files
```

revert

git revert HEAD # undo the previous operation

git revert HEAD~ # undo the previous operation

git revert commit # undo the specified operation

checkout

git checkout -- file # Cancel changes to the file (from the staging area - overwrite the worktree file)

git checkout branch|tag|commit -- file_name # Take the file from the repository and overwrite the current branch

git checkout HEAD~1 [file] # will update working directory to match a commit

git checkout -- . # Extract files from the staging area and overwrite the workspace

git checkout -b gh-pages 0c304c9 # This means that the node with the current branch commit hash value 0c304c9 is divided into a new branch gh-pages and switched to gh-pages

diff

git diff file # View the difference of the specified file

git diff --stat # View simple diff results

git diff # Compare the difference between Worktree and Index

git diff --cached # Compare the difference between Index and HEAD

git diff HEAD # Compare the difference between Worktree and HEAD

git diff branch # Compare the differences between Worktree and branch

git diff branch1 branch2 # Compare the differences between two branches

git diff commit commit # Compare the differences between two commits

git diff master..test # The above command only shows the difference between the two branches

git diff master...test # You want to find the difference between the common parent branch of 'master', 'test' and the 'test' branch, you replace the first two '.' with 3 '.'

stash

git stash # Stash the workspace site (tracked files), and resume work after restoring it later.

git stash list # View saved job sites

git stash apply # restore the job site

git stash drop # delete stash content

git stash pop # Delete stash content directly while restoring

git stash apply stash@{0} # Restore the specified job site, when you have saved more than one job site.

merge

git merge --squash test # Merge and compress, compress the commit on test into one

cherry-pick

git cherry-pick commit # Pick merge, merge commit to current branch

git cherry-pick -n commit # Pick multiple commits, after merging, you can continue to pick the next commit

rebase

git rebase master # Rebase the commits ahead of the master point to the current branch

git rebase --onto master 169a6 # Limit the rollback scope, rebase the commits of the current branch from 169a6 onwards

git rebase --interactive # Interactive mode, modify commit

git rebase --continue # Continue merging after dealing with conflicts

git rebase --skip # skip

git rebase --abort # unmerge

branch

delete

git push origin :branchName # delete remote branch

git push origin --delete new # delete remote branch new

git branch -d branchName # delete the local branch, force delete with -D

git branch -d test # delete the local test branch

git branch -D test # Force delete the local test branch

git remote prune origin # The remote is deleted, and the remote existence can still be seen locally. This command deletes the branch that does not exist in the remote

submit

git push -u origin branchName # Submit the branch to the remote origin host

Pull

git fetch -p #When pulling a remote branch, it is automatically cleaned up. The remote branch has been deleted, and the corresponding branch with the same name still exists locally.

branch merge

git merge branchName # Merge branch - merge branch branchName and current branch

git merge origin/master # Merge remote branch on local branch.

git rebase origin/master # Merge the remote branch on the local branch.

git merge test # Merge the test branch into the current branch

Rename

git branch -m old new #rename branch

Check

git branch # list local branches

git branch -r # list remote branches

git branch -a # list all branches

git branch -v # View the information of the last commit object of each branch

git branch --merge # View the branch that has been merged into the current branch

git branch --no-merge # View the branch merged into the current branch

git remote show origin # You can view the remote address, remote branch

new

git branch test # New test branch

git branch newBrach 3defc69 # Specify the hash 3defc69, and the new branch name is newBrach

git checkout -b newBrach origin/master # After retrieving updates from the remote host, create a new branch based on it

git checkout -b newBrach 3defc69 # Create a newBrach branch with the hash value 3defc69 and switch to this branch

connect

git branch --set-upstream dev origin/dev # Link the local dev branch with the remote dev branch

git branch --set-upstream master origin/next # Manually establish a tracking relationship

branch switching

git checkout test # switch to the test branch

git checkout -b test # New + switch to test branch

git checkout -b test dev # Create a new test branch based on dev and switch

remote

git fetch # fetch retrieves updates from all branches

git fetch origin remotebranch[:localbranch] # Pull the branch from the remote [to the local specified branch]

git merge origin/branch # Merge the specified branch on the remote

git pull origin remotebranch:localbranch # Pull the remote branch to the local branch

git push origin branch # Push the current branch to the specified branch on the remote

git push origin localbranch:remotebranch # Push the specified branch locally to the specified branch on the remote

git push origin :remotebranch # delete the specified remote branch

git checkout -b [--track] test origin/dev # Based on the remote dev branch, create a new local test branch [set tracking at the same time]

submodule

Clone the project and clone the submodule at the same time

git clone <https://github.com/jaywcjlove/handbook.git> --depth=1 --recurse-submodules

Clone the project, then manually clone the submodule subproject

```
git submodule add --force 'warehouse address' 'path'
```

Among them, the warehouse address refers to the submodule warehouse address, and the path refers to the path where the submodule is placed under the current project.

Note: The path cannot end with / (the modification will not take effect), it cannot be the existing directory of the existing project (can not be successfully cloned)

```
git submodule init # Initialize submodule
```

```
git submodule update # Update the submodule (the command must be executed in the root directory)
```

```
git submodule update --init --recursive # The downloaded project has submodule
```

When a submodule is included in the project using git clone, initially, the content of the submodule will not be downloaded automatically. At this time, just execute the following command:

```
git submodule foreach git pull # There are other submodules in submodule One update
```

```
git submodule foreach git pull origin master # submodule update
```

```
git submodule foreach --recursive git submodule init
```

```
git submodule foreach --recursive git submodule update
```

Delete Files

```
git rm -rf node_modules/
```

remote

Git is a distributed code management tool, so it can support multiple warehouses. In git, the warehouse on the server is called remote locally. When personal development, multi-source may not be used much, but multi-source is actually very useful.

```
git remote add origin1 git@github.com:yanhaijing/data.js.git
```

```
git remote # show all sources
```

```
git remote -v # show all sources + details
```

```
git remote rename origin1 origin2 # rename
```

```
git remote rm origin # delete
```

```
git remote show origin # View all information about the specified origin
```

tags

When development reaches a certain stage, tagging a program is a great feature.

```
git tag -a v0.1 -m 'my version 1.4' # New annotated tag
```

```
git push origin --tags # push all branches at once
```


git push origin v1.5 # Push a single tag to the origin source

git tag -v v1.4.2.1 # Verify tags, verify signed tags

git show v1.5 # see the corresponding GPG checkmark

git tag # list existing tags

git tag v0gi.1 # New tag

git checkout tagname # switch to tag

git tag -d v0.1 # delete tags

git push origin :refs/tags/v0.1 # delete remote tags

git pull --all # Get all remote content including tags

git --git-dir=/.git' describe --tags HEAD # View local version information

log log

git config format.pretty oneline #When displaying history, only one line is displayed for each commit

git config color.ui true #colored git output

git log #View recent commit log

git log --pretty=oneline #Single line display commit log

git log --graph --pretty=oneline --abbrev-commit

git log -num #Display the first log (reciprocal)

git reflog #View all operation records of all branches

git log --since=1.day #Commits within one day; you can give various time formats, such as a specific day ("2008-01-15"), or how long ago ("2 years 1 day" 3 minutes ago").

git log --pretty="%h - %s" --author=your name #View your own log

git log -p -2 #Expand two updates to show the content difference of each commit

git log --stat #To quickly see what changes have been made to updates submitted by other collaborators

git log --pretty=format:"%h - %an, %ar : %s"#Customize the record format to be displayed

git log --pretty=format:'%h : %s' --date-order --graph # Topological order display

git log --pretty=format:'%h : %s - %ad' --date=short #date YYYY-MM-DD display

git log HEAD --pretty=format:%s # Only show commits

git config --global format.pretty '%h : %s - %ad' --date=short #date YYYY-MM-DD display Write global configuration

Option Description Option Description

%H The full hash string of the commit object (commit) %ad Author revision date (can be customized with the -date= option)

%h Short hash string of commit object %ar Author revision date, as how long ago

%T The full hash string of the tree object (tree) %cn The name of the committer
%t Short hash string of tree object %ce Submitter's email address
%P full hash string of parent %cd commit date
%p Short hash string of parent object %cr Commit date, as how long ago
%an author's name %s commit description
%ae Author's email address - -

Pretty Formats

rewrite history

git commit --amend # change the last commit

git rebase -i HEAD~3 # 修改最近三次的提交说明，或者其中任意一次

git commit --amend # 保存好了，这些指示很明确地告诉你该干什么

git rebase --continue # 修改提交说明，退出编辑器。

pick f7f3f6d changed my name a bit

pick 310154e updated README formatting and added blame

pick a5f4a0d added cat-file

change to

pick 310154e updated README formatting and added blame

pick f7f3f6d changed my name a bit

删除仓库

cd ..

rm -rf repo.git

Github官方教程

other

git help * # 获取命令的帮助信息

git status # 获取当前的状态，非常有用，因为git会提示接下来的能做的操作

报错问题解决

\1. git fatal: protocol error: bad line length character: No s

解决办法：更换remote地址为 http/https 的

\2. The requested URL returned error: 403 Forbidden while accessing

解决github push错误的办法：

#vim 编辑器打开 当前项目中的config文件

```
vim .git/config
```

#修改

```
[remote "origin"]
```

```
url = https://github.com/jaywcjlove/example.git
```

#为下面代码

```
[remote "origin"]
```

```
url = https://jaywcjlove@github.com/jaywcjlove/example.git
```

\3. git status 显示中文问题

在查看状态的时候 git status 如果是中文就显示下面的情况

```
\344\272\247\345\223\201\351\234\200\346\261\202
```

解决这个问题方法是：

```
git config --global core.quotePath false
```

References

Git官网

Github 15分钟学习Git

Git参考手册

Git简明手册

Git Magic

Git Community Book 中文版

Pro Git

图解Git

git-简明指南

learnGitBranching 在线学习工具

初级教程

廖雪峰的Git教程

蒋鑫老师将带你入github的大门

git详解

oschina教程

How to undo (almost) anything with Git撤销一切，汇总各种回滚撤销的场景，加强学习。

Git 教程 | 菜鸟教程runoob.com

Git 本地仓库和裸仓库

沉浸式学 Git

Git进阶用法，主要是rebase高级用法

All about sudo & su

In 1980, a quantum leap in fine-grained escalation control was conceived by some gentlemen—namely Robert Coggeshall and Cliff Spencer—working at the Department of Computer Science at SUNY/Buffalo. It ran on a VAX-11/750 running 4.1BSD, and they named their software sudo, short for “superuser do”. Originally its only function was to allow temporary privilege escalation to the superuser (or root) account, except using the user’s own password instead of sharing the root account.

- `sudo` stands for "superuser do" or "switch user, do."
 - It allows you to execute a command as the superuser
 - The superuser has full control over the system's hardware and software settings, with no restrictions.
- `su` stands for "substitute user."
 - It allows you to either change the user ID or become the superuser
 - Running `su` without a preceding `sudo` will subsequently prompt you for a password that corresponds to the username you passed to `su`.
 - When you are finished using the account, enter `logout` or `exit` or press `Ctrl` + `D` to return to the account in which you started (Figure 1).
 - Should you become confused about which account you are in, you can type the `whoami` command to orient yourself.
 - If you do not specify a username after the `su` command, the system logs you into the root account.
 - Entering `su root` will only change the account, and not the environment. The same problem also exists with other accounts, and you can solve it by entering: `su - [USER NAME]`. The `-l` or `--login` options have the same function.
 - `su -c` allows you to pass a command to the shell and run it as root, such as when you enter commands like `su -c "less /var/log/messages"`. The system responds by displaying the file in the `less` viewer. When you press the `q` key to quit the viewer, you return immediately to the original account, having spent the minimal time possible as root. Quotation marks mean the command should be read as a continuous option of the `su -c` command. Without the quotation marks, `su` recognizes that `less` is a command but, expecting its usual syntax, wrongly interprets the file path as a username.

Figure 3: su needs quotation marks to read a literal command.

```
bb@nanday:~$ su -c less /var/log/messages
No passwd entry for user '/var/log/messages'
```

Figure 1: Using su to switch accounts is much quicker than logging in and out.

```
bb@nanday:~$ su -
Password:
root@nanday:~# exit
logout
bb@nanday:~$
```

Note: When you use the basic `su` command, you change accounts but do not completely change your environment. To be specific, only the `$HOME`, `$SHELL`, `$USER`, `$LOGNAME`, `$PATH`, and `$IFS` environment variables are reset. Depending on how `su` was compiled, `$TERM`, `$COLORTERM`, `$DISPLAY`, and `$XAUTHORITY` may also be reset.

```
HOME=/home/cory
SHELL=/bin/bash
USER=cory
LOGNAME=cory
PATH=/home/cory/.local/bin:/home/cory/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
TERM=xterm-256color
COLORTERM=truecolor
DISPLAY=:0
XAUTHORITY=/home/cory/.Xauthority
```

The options `-m`, `-p`, or `--preserve-environment` will keep your original environment. You can also specify the shell to use when you switch accounts with `-s` or `--shell [SHELL]` (e.g., if you want to see whether a script written for Bash will run in an alternative shell such as Zsh).

All this tinkering with environments can be confusing, which is why many users stay with `su -`. However, this is the least safe alternative: you are better off running the `env` command if you become confused. The long output that `env` provides contains entries, such as the home directory, that will show just what environment you are using.

Many users enter `su -`, change to root, and then enter a command. This habit is methodical, but it also increases the amount of time your system is vulnerable.

The Sudo Command

Functionally, `sudo` is the equivalent of `su -c`, designed to run a single command as root and then return to the original account. However, whereas `su` requires the root password, `sudo` can be configured several ways, depending on how a particular distribution decides it is most secure.

Some Linux distributions, such as Ubuntu, let users enter their own passwords to use `sudo`. This practice lets a user perform privileged operations without the root password. On the other hand, you control which tasks the user can perform, and letting users authenticate with their own password minimizes the number of users who need access to the root password. Other distributions, such as openSUSE, requires users to enter the root password with `sudo`.

Most users know `sudo` as the equivalent of the children's magic word "please." That is, to run a command with root privileges, you add `sudo` to the start. For example, to run the shutdown command to stop the system, you would type `sudo shutdown` and enter the appropriate password (typically your password or the root password, depending on your `sudo` configuration).

Your expanded privileges can be set to last for a time after you enter the first command - typically, five minutes. Until they expire, you can enter additional commands prefaced by `sudo` without entering a password again. You can extend this time by another 15 minutes using the `-v` option. Conversely, if you do not need the additional time, run `sudo -K` to remove your privileges. Less drastically, `sudo -k [COMMAND]` removes your current privilege for a specific command you are authorized to run with `sudo`. To restore the privileges, you need to run `sudo` again.

As with `su -`, `sudo` offers some control over the environment in which it runs. Using the `-U` option with `sudo`, you can run a command as a user other than root. You can also use `-E` to run commands in your current environment or `-H` to run them from your current home directory.

If you have trouble running `sudo`, start by running `sudo -l` (Figure 4). This command shows the paths to commands that you can run from the current account, as well as the set of commands themselves. In Ubuntu-derived distros, You are probably authorized to run all commands as root, but on a custom configuration, your choices may be more limited. Note that if you do not see a list, but are only offered three attempts to log in, then `sudo` is not configured on your system.

Figure 4: Query `/etc/sudoers` to learn what commands you can run.

```
bb@bb ~ $ sudo -l
Matching Defaults entries for bb on this host:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User bb may run the following commands on this host:
    (ALL : ALL) ALL
    (root)_NOPASSWD: /usr/lib/linuxmint/mintUpdate/checkAPT.py
```

You might think to run `sudo` itself with the full path, `/usr/bin/sudo /bin/bash`, to avoid the possibility of using a fake `sudo`.

sudoers

The `/etc/sudoers` file contains information on which users can use `sudo` and for what purposes. The `sudoers` file is a plaintext file, but you should never open it directly from a text editor. Instead, run either the command `sudo -e sudoedit` or the command `visudo /etc/sudoers`. Both commands lock the original file and open a temporary copy of `sudoers` in the default text editor. If you prefer, you can replace the default editor by running `EDITOR=[EDITOR] /usr/sbin/visudo`.

When you are finished, `visudo` checks for errors, giving you the chance to correct them or go ahead and save to the original file. Be careful before you save - syntactical errors can prevent `sudo` from working properly, or even at all.

The top of the `sudoers` file sets aliases for advanced configuration. Aliases can be used for such purposes as creating comma-separated lists of users or commands to simplify configuration. For example, if you want to restrict who can power off the system or network, add the following line to create the command alias `SHUTDOWN`:

```
Cmdnd_Alias    SHUTDOWN = /sbin/halt, /sbin/shutdown, /sbin/reboot, /sbin/poweroff
```

Once the alias is defined, you can give users the right to run all four commands simply by referencing the `SHUTDOWN` alias. In the same way, you could define a group of users called `ADMINS`, all of whom can run the same commands.

Regardless of whether you specify aliases, you can assign privileges with single-line entries. Depending on the distribution, the assignment of various privileges may be organized by commented lines prefaced with the hash (`#`) symbol. Be sure that the lines that assign privilege are below the list of uncommented aliases.

The simplest form of privilege assignments is `[NAME] ALL = ALL`. For example, `bb ALL = ALL` allows user `bb` to run any command from any terminal.

More restrictively, the line could be read as follows:

```
bb /sbin/halt/, /sbin/shutdown, /sbin/reboot, /sbin/poweroff=ALL
```

Or, if the `SHUTDOWN` alias suggested above was defined at the top of the `sudoers` file, the line could be:

```
bb SHUTDOWN=ALL
```

The `sudoers` file can also contain fields to adjust other behaviors. The most useful fields are `passwd_tries`, which sets the number of attempts to log in to `sudo`; `passwd_timeout`, which sets the length of time that a login lasts; and `editor`, which sets the editors you can use with `sudo`. For a complete list of these fields, see the `sudoers` man page.

Avoiding Self-Sabotage

Like much of Linux, `su` and `sudo` can be as simple or as complex as you choose. Most popular uses of `sudo`, in particular, are extremely basic, and by copying them, you can quickly get up to speed. However, when you use `su` and `sudo`, be careful that you do not undermine their purpose. The entire point of both commands is to increase security by minimizing the time you run as root. Both commands can reduce your time as root to the bare minimum, but that doesn't mean you can relax other precautions.

Specifically, avoid using `su` to become root and then keeping a terminal open and forgotten on some overlooked virtual workspace. Similarly, reduce the time that a successful `sudo` login lasts to the minimum. Consult system logs to ensure that the powerful `su` and `sudo` commands are only used for authorized activities.

The Administrator Sees Everything

The system will log unsuccessful calls to `su` and `sudo`. If you type `su` and enter the wrong password, an entry like the following will appear in the logfile (`messages`, `syslog`, or `auth.log` below the `/var/log/directory`):

```
Nov 23 08:14:08 server su: FAILED SU (to root) esser on /dev/pts/3
```

Unsuccessful calls to `sudo` are also logged. Typically, you will see entries in the `/var/log/messages` file, although Fedora Linux writes the entries to `/var/log/secure` and only logs the errors in `messages`:

```
Sep 11 21:10:47 huhnix sudo: huhn : TTY=pts/6 ; PWD=/home/huhn ; USER=root ;  
COMMAND=/usr/bin/less  
/var/log/messages
```

The `sudo` command differs from `su` in that the logfile also reveals which program a user ran with root privileges, but that's not all. If you attempt to run a program without the necessary privileges, don't be surprised if the administrator knows exactly what you have been up to and sends you a friendly reminder about unsuccessful "break-in attempts." Users also learn that they are not allowed to run `sudo` thanks to a console message:

```
$ sudo less /var/log/messages  
Password:  
peggy is not in the sudoers file.  
This incident will be reported.
```

In plain English, this message means that the root user will receive a message by email that shows exactly when "peggy" failed to run `sudo`. The message is tagged as "SECURITY information for huhnix." If you want to find out which commands you are allowed to run under `sudo`, you can type `sudo -l` for a list of commands allowed through the `sudo` configuration.

sudo command	Explanation
<code>sudo -i</code>	Begin a shell session as the system's 'root' user.
<code>sudo -B</code>	Execute instructions in the background.
<code>sudo -h</code>	Show help (from man page).
<code>sudo -K</code>	When given 'sudo' privileges for a certain duration of time, <code>sudo -K</code> removes those cached credentials entirely, so that the next time you run a command with 'sudo' privileges, you will have to enter a password.
<code>sudo -k [COMMAND]</code>	Removes your current privilege for a specific command you are authorized to run with <code>sudo</code> .
<code>sudo -V</code>	Print the <code>sudo</code> version string as well as the version string of the security policy plugin and any I/O plugins.
<code>sudo -v</code>	Update the user's cached credentials, authenticating the user if necessary. For the sudoers plugin, this extends the sudo timeout for another 15 minutes by default, but does not run a command. Not all security policies support cached credentials.
<code>sudo -l</code> or <code>sudo -ll</code>	List user's privileges or check a specific command; use twice for longer format.
<code>sudo -s [SHELL]</code>	Execute the specified shell.

Assigning Access Permissions to Users and Groups

Users and groups are concepts central to multiuser operating systems. Assigning a name to a user account allows users to log in separately, set up their own environments, and control access to their private files by assigning permissions.

Strategies for applying users and groups have changed through the years, but essentially, a group is a collection of users typically assigned access to a collection of resources associated with a specific function or profile. For instance, the financial group might be assigned access to a common directory with financial documents or access to a printer located in the financial office.

On home systems with only a few users, groups sometimes matter so little that users mostly ignore them. In fact, you can sometimes hear suggestions that groups are obsolete and should be eliminated. However, on networks, groups are a means of exercising the security principle of **least privilege: restricting access to data and functions only to those who require them**. Modern Linux systems have GUI-based utilities for managing users and groups, but many experienced users still prefer the swift and decisive Bash commands.

Files for Users and Groups

Users are defined on one line in `/etc/passwd` and a corresponding line in `/etc/shadow`; memberships in groups are defined in `/etc/group`. Each user is assigned a unique identifier, called the user ID (UID), and each group is assigned a group identifier, referred to as the GID. **Valid UIDs and GIDs are integers from zero to $2^{32} - 1$, although the maximum recommended is 65535.**

UIDs between 0 and 999 are generally reserved for system accounts. The root account typically has a UID of 0 and belongs to a group also known as root, which is assigned a GID of 0. By default, most Linux distributions number user accounts from 1000, which is assigned to the user account during installation.

The `/etc/passwd` file is readable by every user of the machine. Because allowing users to view even encrypted passwords is a security hazard, passwords were long ago removed from `/etc/passwd` to `/etc/shadow`, which only the superuser can view. In fact, with the introduction of `systemd`, `/etc/passwd` has become even more limited. Each account continues to be defined in six colon-separated fields (Figure 1), starting on the left with the username. The second field is now always marked with an `x`, which originally indicated that the account could be used to log in, but now it seems to indicate that the field is governed by `systemd`.

The third and fourth fields are still the UID and the GID of the group to which the user belongs, but the fifth field, which once stored additional information about a user, such as a full name and phone number, now either points to a subdirectory of `/run/systemd` or of `/var`, which provides resources for `systemd`. The sixth field, can still list the user's shell, but `/sbin/nologin` is more likely to be entered than in the pre-`systemd` days.

Figure 1: The `/etc/passwd` file under `systemd`.

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
```

The `/etc/shadow` file has preserved its original functionality (Figure 2). The first field in each line is the username, and the second is the encrypted password - or a placeholder if the user cannot log in. The third through fifth fields are used for controlling passwords - showing the age of the password, the minimum age before the password can be changed, and the maximum time before a password must be changed. The sixth field is supposed to define the number of days before a password expires that the user will receive a warning, the seventh field defines the number of days after expiry that the account will be disabled, and the eighth field is left blank for future purposes.

Figure 2: `/etc/shadow` contains encrypted passwords where they exist, but many of its fields are no longer used by many users.

```
daemon:*:15434:0:99999:7:::
bin:*:15434:0:99999:7:::
sys:*:15434:0:99999:7:::
sync:*:15434:0:99999:7:::
games:*:15434:0:99999:7:::
```

The use of `/etc/group` has also changed over the years. Each group is defined in a single line of four fields (Figure 3). The first is the group's name, which is usually self-explanatory. The name is followed by a field for a group password, which these days is almost always marked by an `x`, because the custom is to rely on user passwords. The third field is the GID, and the fourth is a comma-separated list of users.

When you install most distributions, the ordinary user account created at installation is added automatically to the groups that an average desktop user might need, such as **cdrom** or **sudo**. When you add a user account, the user is typically added to the same groups assigned to the user account created at installation. You can also add a user to a group directly by opening `/etc/group` in a text editor as root.

Figure 3: `/etc/group` remains useful, but it is also a repository of obsolete groups.

```
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:logcheck
```

Adding, Locking, and Removing

The command for adding users is run from the root account:

```
adduser OPTIONS USER
```

The `adduser` command calls the **useradd** utility. You can also call **useradd** directly from the command line on most Linux systems, but `adduser` is a bit more user friendly.

By default, the user and a group with the same name are created. You are asked to type the password for the new account, and a home directory is created with the same name as the account under `/home` (Figure 4). However, you can use options to modify all these defaults, adding or changing the default groups or the location of the home directory. You can also set the number of days before the account expires or set the account to use a shell other than Bash. To save time, you can also edit `/etc/default/useradd` to set the default information created for new accounts.

Figure 4: The `adduser` command sets up a basic account.

```
root@nanday:/etc/default# adduser jack
Adding user `jack' ...
Adding new group `jack' (1002) ...
Adding new user `jack' (1004) with group `jack' ...
Creating home directory `/home/jack' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for jack
```

```
Enter the new value, or press ENTER for the default
```

```
Full Name []:
```

```
Room Number []:
```

```
Work Phone []:
```

```
Home Phone []:
```

```
Other []:
```

```
Is the information correct? [Y/n]
```

After you enter the password, you can add user information such as the **Full Name**, **Room Number**, **Work Phone**, and **Home Phone**, but many users simply press the Enter key to bypass these fields. Later, you can change the password with `passwd USER`, authenticating with the current password before changing it.

If you want to disable a user account rather than delete it, the easiest solution is the command

```
passwd --expire 1 USER
```

This command sets an expiry date in the past, automatically making the account unavailable. It is preferable to `passwd -l`, because it not only prevents a normal login, but also a login by **ssh**. When you want to restore the account, use `passwd -u USER`.

When you decide to delete an account, first transfer any information you want to save from the home directory. To delete a user account, log in as root and type:

```
userdel USER
```

If you no longer need the home directory, add the `-r` option. However, you can also use the `-f` option to delete the account, even if the user is logged in. Note that the command gives no output of its progress; the only sign that the account has been deleted is when you return to the command line.

Similar commands, `groupadd` and `groupdel`, are available for editing groups. Generally the only groups you will want to edit are those created for users. For predefined groups that give access to hardware or functionality, it is usually better to just take all the users out of an unused group rather than deleting the group itself.

If you do delete a group, be aware that deleting the group does not change permissions on any files. Before deleting a group, locate any files that the group owns with the command

```
find / -gid GID
```

and change the group permissions.

Managing Users and Groups

Before editing a user or group, make sure that no one who is logged in will be affected by typing the `w` or `who` command. If you want more detailed information, use `finger`, adding a user after the command for specific information.

As you edit an account,

```
usermod OPTIONS USER
```

should take care of most circumstances. For example, you can change the name of an account with the option `-I NEWNAME`, the home with `-d DIRECTORY`, the expiry date with `-e DATE`, or the number of days after the expiry date that the account is disabled with `-f NUMBER`; better yet, combine the two options.

The `groupmod` command performs most of the functions of `usermod`, but for groups (e.g., change the group name with `-n` or the GID with `-g`). If you are using a `chroot` jail (an isolated directory structure), you can also use `-R` to create a new `/etc/group` using the configuration of the jail. However, many users will find `usermod` more handy than `groupmod`.

Permissions

The whole purpose of users and groups is to have a way to assign permissions. Granular access privileges for files and directories make Linux a safe operating system.

The root user is subject to no restrictions, and this includes assigning read, write, and execute permissions to other users throughout the system. If you are the owner of a file or directory, you can grant access to these resources to other accounts. If you are also a member of a specific group, you can modify the group ownership of files and folders for more granular permission assignments to files.

For every file, (and thus for directories, device files, etc.), Linux defines who may read, write, and execute that file. Also, every file belongs to an owner and to a group. The following three permissions are assigned separately for owners, groups, and other users:

- Read permission (`r` flag): Users can display the contents of a file or folder on screen, copy the file, and do a few other things. Directories should additionally have the `x` flag (see below) to allow users to change to that folder; otherwise, only a list of files can be displayed.
- Write permission (`w` flag): Users can change files and directories and store their changes. Write permission also includes the ability to delete.
- Execute permission (`x` flag): For programs, this means that the user is permitted to run the program. Execute permission for a directory means that the user is permitted to change to the directory (the user additionally needs read permission to be able to view the folder content).

To discover the permissions for a file at the command line, you can simply set the `-l` flag with the `ls` command (Figure 5). Permissions are indicated by the letters `r` (for read), `w` (for write), and `x` (for execute). In the `ls` output, note the three sets of `r`, `w`, `x` (or `-`) at the beginning of the file name entry. The first block shows the permissions for the owner, the second block refers to the group, and the third block refers to all users.

- Folders are indicated by a `d` (for "directory") at the start of the list.
- Regular files are indicated by a single dash (`-`).
- Symlinks are indicated by an `l` (for link).
- Block devices like `/dev/sda1` are indicated by a `b` (for block).
- Character devices (e.g., `/dev/tty1`) are indicated by a `c`.

Figure 5: Listing permissions at the command line.

```
magazine@MacBuntu:/$ ls -lr
total 96
lrwxrwxrwx  1 root root    30 2010-12-25 08:23 vmlinuz.old ->
boot/vmlinuz.old...
lrwxrwxrwx  1 root root    30 2011-01-27 16:14 vmlinuz.old -> boot/vmlinuz-...
drwxr-xr-x 15 root root 4096 2010-10-07 12:38 var
drwxr-xr-x 12 root root 4096 2010-04-15 12:35 usr
drwxrwxrwt 15 root root 4096 2011-02-04 11:09 tmp
drwxrwxr-x 12 root root    0 2011-02-04 11:08 sys
drwxr-xr-x  3 root root 4096 2010-09-09 17:08 srv
```

Special Permissions

Linux has two special permissions:

- the **s** bit (also known as the setuid/setgid bit)
- the **t** bit (also known as the sticky bit)

Both replace the **x** in the **rwX** block of three. The **s** is commonly seen with executable files, whereas the **t** bit is more common with directories.

The setuid/setgid (set user ID/set group ID) bit executes a program with the permissions of the user or group, no matter who runs the program. In this way, nonprivileged users can access resources they would not normally be able to access. Although this is a potential security risk, the **s** bit has its uses. Many programs, including **su**, **sudo**, **mount**, or **passwd** rely on the **s** bit (Listing 1).

Listing 1: Programs That Use the setuid/setgid Bit

```
-rwsr-xr-x 1 root root 31124 Jul 31 15:55 /bin/su
-rwsr-xr-x 1 root root 123448 Jun 22 18:14 /usr/bin/sudo
-rwsr-xr-x 1 root root 72188 Oct 22 23:54 /bin/mount
-rwsr-xr-x 1 root root 41292 Jul 31 15:55 /usr/bin/passwd
```

The **passwd** program, for example, modifies passwords, accessing the **/etc/shadow** file in the process of entering the new password. By default, the file is protected against write access by nonprivileged users and reserved for use by the administrator to prevent just anybody having the ability to manipulate the passwords. The **s** bit executes the **passwd** program as the root user and enters the new password in **/etc/shadow** "on behalf" of root.

The other special permission, the **t** bit, commonly occurs in shared directories (read, write, execute permissions for all) in place of the execute flag to ensure that users are only allowed to modify - and thus delete - their own data. The sticky bit is typically set for **/tmp** (Figure 5). This stores temporary files for multiple users.

If everybody had the right to read, write, and execute these files, in theory, everybody would be able to clean up the system and delete arbitrary data. The **t** bit ensures that users can only delete their own files (or those files for which they have write permission). The exception to this rule is that the owner of the folder with the sticky bit is allowed to delete within that folder.

Modifying Permissions

The `chmod` program lets you modify file and directory permissions, assuming you are the owner or the system administrator. `chmod` lets you set the permissions using either letters or numbers.

If you are using letters

- `u` stands for user (owner)
- `g` stands for group
- `o` stands for others (all other users)

As I described previously

- `r` stands for read
- `w` stands for write
- `x` stands for execute
- `s` stands for the setuid/setgid bit
- `t` stands for the sticky bit

A combination of these letters (without spaces!) with plus, minus, and equals signs tells `chmod` to add, remove, or assign these permissions (Table 1). To give a group read and write permissions for a file, just type `chmod g+rw file`. Removing permissions follows the same pattern: The `chmod o-rwx file` command removes all permissions for all user who are neither the owner nor members in the owner group. You could combine these two commands thus:

```
chmod g+rw,o-rwx file
```

(See the "Permissions and Priorities" box for the hierarchy of permissions.)

Permissions and Priorities		
Permissions for the user, group, and all others have different priorities. If you are the owner of a file, permissions for the owner apply (the first block of three letters). If you're not the owner but belong to the group, the second block applies. If you're neither the owner nor a group member, the third set of permissions apply.		

Table 1: Permissions

Octal	Binary	Letters
0	000	---
1	001	--X
2	010	-W-
3	011 (= 2 + 1)	-WX
4	100	r--
5 (= 4 + 1)	101 (= 4 + 1)	r-X
6 (= 4 + 2)	110 (= 4 + 2)	rw-
7 (= 4 + 2 + 1)	111 (= 4 + 2 + 1)	rwX

An equals sign lets you assign precisely the permissions specified at the command line. For example,

```
chmod ugo=rwx directory
```

gives the owner, group members, and all other users read, write, and execute permissions for the directory. Instead of `ugo`, you could alternatively use a (for all) to assign user, group, and other permissions.

The `chmod` program also understands numbers. Instead of specifying the permissions with letters, you can pass in three- or four-digit octal numbers. The octal number is an ingenious shorthand for referring to a binary number that spells out the `rwX` permission bits (see Table 1). Calculate the numbers as follows:

- `4` stands for read permission
- `2` stands for write permission
- `1` stands for execute permission

The first number refers to the owner, the second number to the group, and the third to all others.

On this basis, you can see, for example, `644` would mean `u=rw,go=r` (resulting in `rw-r--r--`), or `777` would be `a=rwx` (resulting in `rxwxrwxrwx`).

To set the `s` or `t` bit, you need to add a fourth number at the start of the block of three. The number `4` represents the `s` bit for the owner (setuid), `2` sets the `s` bit for the group (setgid), and `1` sets the `t` bit. Listing 2 gives an example.

Listing 2: Setting the s Bit by Number

```
$ ls -l script.sh
-rw-r--r-- 1 heike heike 3191789 Oct 6 05:01 script.sh
$ chmod 4755 script.sh
$ ls -l script.sh
-rwsr-xr-x 1 heike heike 3191789 Oct 6 05:01 script.sh
```

Changing Group Memberships

To change group membership for files and directories, you can use the `chgrp` tool. As a "normal" user, you are allowed to assign your own files to a specific group; however, this assumes that you are a member of the group. The root user, as always, has no restrictions.

The following command tells you your own group memberships:

```
$ groups
petronella adm dialout fax cdrom tape audio dip video plugdev fuse lpadmin
netdev admin sambashare
```

In this case, the user called `petronella` may change access to her own files for members of the groups `petronella`, `adm`, `dialout`, `fax`, `cdrom`, and so on. The `chgrp` command first expects information about the new group and then the name of the file or directory. To assign a file to the `audio` group, just type:

```
chgrp audio FILE_NAME
```


Changing Owners and Groups

On a Linux system, the system administrator is allowed to assign new owners and new groups to files and directories. To give a file to user `petronella`, simply use the `chown` command:

```
chown petronella FILE_NAME
```

Also, you can define a new group in the same command. To do so, add the name of the group after a colon:

```
chown petronella:audio FILE_NAME
```

The file now belongs to user `petronella` and group `audio`.

`chown` is mainly used by the root user, however, an ordinary user can use it for certain limited tasks, such as changing the group membership for a file the user owns to a group to which the user belongs.

Across the Board

All three tools - `chmod`, `chgrp`, and `chown` - support the `-R` option for recursive actions. If you want members of the `video` group to access a directory and the files it contains, just type:

```
chgrp -R video DIRECTORY
```

The `-R` option can also save you some typing in combination with the `chmod` command. To remove read, write, and execute permissions from this folder for all users who are **not** the owner or members of the `video` group, just type:

```
chmod -R o-rwx DIRECTORY
```

Be careful when you run recursive commands that remove the execute flag. If you mistakenly type `a-x` instead of `o-x`, you will lock yourself out:

`chmod` will remove execute permissions from the parent directory and your ability to change the directory and modify files (Listing 3). The use of the `find` command can help you avoid this kind of dilemma (Listing 4). The `find` command first discovers the files (`-type f`) in the `test` directory (and possible subfolders) and then runs `chmod` against them, ignoring the directory itself.

Listing 3: Oops ... Locked Out!

```
$ ls -l test
total 0
-rwxr-xr-x 1 petronella petronella 0 Nov  4 12:12 bar
-rwxr-xr-x 1 petronella petronella 0 Nov  4 12:12 foo
$ chmod -R a-x test
chmod: cannot access `test/bar': Permission denied
chmod: cannot access `test/foo': Permission denied
```

Listing 4: Using the find Command

```
$ find test -type f -exec chmod a-x \{\} +
$ ls -l test
total 0
-rw-r--r-- 1 petronella petronella 0 Nov  4 12:12 bar
-rw-r--r-- 1 petronella petronella 0 Nov  4 12:12 foo
```

To use the `-R` parameter with the `chown` program, you would enter the following command to hand over the home directory and all the files in it (including the hidden configuration files) to user `pooh`:

```
chown -R pooh /home/pooh
```

From the Beginning

The `umask` program specifies the default permissions assigned to newly created files and directories. Typing the `umask` command without setting any parameters reveals the current setting:

```
$ umask
0022
```

The four-digit octal number that is returned specifies what to subtract from the default values (`0666` for files, `0777` for directories). In other words, new files are assigned `0644` (`rw-r--r--`), and new folders are assigned `0755` (`rw-r-xr-x`) when they are created.

To change the `umask`, enter the file and specify the new value at the command line:

```
umask 0077
```

This entry means that new files and directories are only available to their owner. To make new files writable for group members, you can choose `umask 0002` instead.

The `umask` you assign in this way is valid for the current shell, but you can add an entry to your Bash configuration file `~/.bashrc` to make the change permanent. Don't forget to run `source ~/.bashrc` to reload the Bash configuration file.

To modify the `umask` for the system, you will need to add a global entry to the `/etc/profile` file, and you will need to work as root to edit it.

Calendar command

```
cal
# ^^^ Displays calendar for current month
cal -y
# ^^^ Displays calendar for current year
ncal
# ^^^ Displays calendar for current month, with the rows and columns reversed
ncal -h
# ^^^ Turns date highlight off
cal -m1
# ^^^ Displays first month of the current year
ncal -m1
```

```

# ^^^ Displays first month of the current year, with the rows and columns
reversed
ncal -w
# ^^^ Turns date highlight back on
cal -3
# ^^^ Displays previous month, the current month, and the next month
ncal -3
# ^^^ The same as the previous command, but with the rows and columns reversed
cal -A2
# ^^^ Displays the current month, the next month, and the month after next
month
ncal -A2
# ^^^ The same as the previous command, but with the rows and columns reversed
cal -B2
# ^^^ Displays the two months prior to the current month, as well as the
current month
ncal -B2
# ^^^ The same as the previous command, but with the rows and columns reversed
cal -B4 -A3
# ^^^ Displays current month, four months prior, and three months after the
current month
cal 2021
# ^^^ Displays the current calendar year
cal -d 2016-3
# ^^^ Displays a specific month from a specific calendar year
paste <(cal 2020) <(cal 2021)
# ^^^ Displays two specific calendar years side by side

```

History command

`bash_history` file is in your home directory

```

history                # show command history
history 20             # Display last 20 commands
history -c             # clear command history
history | grep chpasswd # retrieves all instances of specified
                        # string from command history

[Up arrow]           # Previous command
Ctrl + R              # Reverse search
!ls                   # Run last ls command
!!                    # Previous command
!45                   # Run the command with the ID number 45
!-5                   # Revert back to the fifth previous command relative
                        # to current command ID
!1997:p               # Prints command before executing
^string1^string2^     # Replace string1 with string2 in previous command
fc 477 487            # (file compare) Displays specific range of commands
in default text editor

```

List command

```
ls
# ^^^ list
ls -l
# ^^^ long listing
ls -a
# ^^^ show hidden files
ls -h
# ^^^ human-readable
ls -r
# ^^^ reverse order
ls -t
# ^^^ sort files by date from newest to oldest
ls -S
# ^^^ sort by decreasing size
ls -i
# ^^^ show inodes
ls -1
# ^^^ show 1 item per line
ls -d
# ^^^ list directories themselves, not their contents
ls -X
# ^^^ sort alphabetically by entry extension
```

Change directory command

```
cd
# ^^^ change directory
cd .
# ^^^ The period represents the current working directory
cd OR cd ~
# ^^^ Navigate to the home directory of the current user (don't use the vertical pipes here)
cd /
# ^^^ Navigate to the root directory
cd -
# ^^^ Navigate to the previously chosen directory
cd ..
# ^^^ Go back one directory
cd ../../
# ^^^ Go back two directories
cd ../../drop/into/directory
# ^^^ Go back two directories and drop into desired directory
cd /path/to/directory
# ^^^ Navigate to desired directory using absolute path
cd path/to/directory
# ^^^ Navigate to desired directory using relative path
```

Print working directory

```
pwd
# ^^^ print working directory
```

Make directory command

```
mkdir
# ^^^ make directory
mkdir -p foo/bar/baz
# ^^^ Create nested directories, including "parents" and "children"
mkdir -p foo/{bar,baz}
# ^^^ Create parent directory with two subdirectories
mkdir -p foo/{bar,baz/{zip,zap}}
# ^^^ Create grandparent directory, two child directories, and two grandchild
directories
```

Copy command

```
cp ~/Desktop/foo.txt ~/Downloads/foo.txt
# ^^^ copy a file
cp -r ~/Desktop/cruise_pics/ ~/Pictures
# ^^^ copy a directory
cp -i ~/Desktop/foo.txt ~/Documents/foo.txt
# ^^^ To create a copy but ask to overwrite if the destination file already
exists
cp foo.txt{,."$(date +%Y%m%d-%H%M%S)"}
# ^^^ To create a backup file with date
cp path/to/source_file.ext path/to/target_file.ext
# ^^^ Copy a file to another location
cp path/to/source_file.ext path/to/target_parent_directory
# ^^^ Copy a file into another directory, keeping the filename
cp -R path/to/source_directory path/to/target_directory
# ^^^ Recursively copy a directory's contents to another location (if the
destination exists, the directory is copied inside it)
cp -vR path/to/source_directory path/to/target_directory
# ^^^ Copy a directory recursively, in verbose mode (shows files as they are
copied)
cp -i *.txt path/to/target_directory
# ^^^ Copy text files to another location, in interactive mode (prompts user
before overwriting)
cp -L link path/to/target_directory
# ^^^ Dereference symbolic links before copying
```

Finding files

The *find* command lets you search for files by name or part of the name. By default, *find* searches recursively, meaning it looks for files through the entire directory tree.

```
find /home/foo -name "Lin*"
# ^^^ Search for files whose names start with Lin in the foo directory and its
subdirectories
# Remember that the find command is case sensitive
find home/user -iname "Lin"
```

```

# ^^^ The -iname switch tells the find command to ignore case
find /usr /home /tmp -name "*.bin"
# ^^^ Search through the /usr, /home, and /tmp directories to look for all bin
files
find /usr /home /tmp -name "*.bin" 2> /dev/null
# ^^^ Avoid cluttered output by sending all error messages to the null
pseudodevice file
find /photos -iname "*.NEF" -and -size +7M
# ^^^ Search for .NEF files that are larger than 7MB
find /downloads -size +7M ! -iname ".NEF"
# ^^^ Search for files that are larger than 7MB but are not .NEF files
find /downloads -size +7M -or -iname "*.NEF"
# ^^^ Search for files that match either of the specified criteria
find . -user root
# ^^^ Search for files by owner (user)
find . -group www
# ^^^ Search for files by owner (group)
# -type options: f(regular file), d (directory), l (symbolic links), and a few
others
find berlin/ -type d
# ^^^ Search for the directory of photos from Berlin

# time options: -mmin (last time modified in minutes), -amin (last time accessed
in minutes), -cmin (last time changed in minutes), -mtime (last time modified in
hours, -atime (last time accessed in hours), -ctime (last time changed in hours)

find /photos -mmin -10 -name "*.NEF"
# ^^^ Search for photos that were modified 10 minutes ago
find /home/user -name "Lin*" -fprint search_results.txt
# ^^^ Print the search results to a file

```

The *locate* command searches the database of filenames, which is updated daily.

```

locate backup.sh
# ^^^ locate file
locate -i backup.sh
# ^^^ locate file, while ignoring case
locate "*.jpg"
# ^^^ locate files by extension
locate "*.jpg" -n 5
# ^^^ locate a specific number of results
updatedb
# ^^^ manually update the file name database

```

The *whereis* command returns paths to binary, source, and man pages for an executable.

```

whereis rawstudio # returns binary, source, and man page info for the Rawstudio
application
# To search only binaries, use -b option
# To search only man pages, use -m option
# To search only source files, use -s option

```

The *which* command

The *which* command tells you which version of a command will run. For example, the *which soffice* command returns the `/usr/bin/soffice` path. If you want to find all the locations of the command, you can use the `-a` option:

```
which -a soffice
```

remove file(s)

```
rm testfile.txt
# ^^^ Remove a file by passing the name of the file or path of the file to the rm
command
rm filename1 filename2 filename3
# ^^^ Removes multiple files from the current directory
rm -i filename(s)
# ^^^ Confirms the removal of each file before deleting it
rm -I filename(s)
# ^^^ Prompts once before removing more than three files, or when removing
recursively
rm -f filename(s)
# ^^^ Remove files without prompting (force), even if the files are write-
protected
rm -v filename(s)
# ^^^ Explains what is going on (verbose mode)
rm -d dirname(s) || rmdir dirname(s)
# ^^^ Remove empty directories
rm -r dirname(s)
# ^^^ Remove non-empty directories recursively, and prompts for files that are
write-protected
```

File compression

Packages to Install

```
$ sudo apt install tar gzip bzip2 xz-utils zip unzip rar unrar p7zip-full p7zip-rar
```

Supported file extensions

Archive Type	File extensions supported
a normal tar archive	.tar
a gzipped tar archive	.tgz, tar.gz, taz, tpz
a bzip2 tar archive	tar.bzip2, .tb2, .tbz, .tbz2, .tz2
an lzip tar archive	.tar.lz
an lzma tar archive	.tar.lzma, .tlz
an lzop tar archive	.tar.lzo

Short Option	Long Option	Algorithm
<code>z</code>	<code>--gzip</code>	gzip
<code>j</code>	<code>--bzip2</code>	bzip2
<code>J</code>	<code>--xz</code>	xz
<code>z</code>	<code>--compress</code>	LZA (compress)
	<code>--lzip</code>	lzip
	<code>--lzma</code>	lzma
	<code>--zstd</code>	zstd

These are not your only options, there's more. `tar` accepts `-I` to invoke any third party compression utility.

Short Option	Algorithm
<code>-Iplzip</code>	Parallel lzip
<code>-Ipigz</code>	Parallel gzip
<code>-Ipxz</code>	Parallel XZ (LZMA)
<p>The above arguments will only work if you actually have <code>plzip</code> and <code>pigz</code> installed. Also note that you will have to have <code>c</code> or <code>x</code> before and <code>-f</code> after <code>-I</code> when you use <code>-I</code>. Example: <code>tar c -I"pigz -9" -f archive.tar.gz folder/</code></p>	

Algorithm	Time	Size	Command	Parameters	Comment
none	0m0.934s	939M	tar	cf	tar itself is an <i>archiving</i> tool, you do not <i>need</i> to compress the archives.
gzip	0m23.502s	177M	gzip	cfz	
gzip	0m3.132s	177M	pigz	c -Ipigz -f	Parallel gzip using pigz 2.4.
bzip2	1m0.798s	134M	bzip2	cfj	Standard bzip2 will only use <i>one</i> core (at 100%)
bzip2	0m9.091s	135M	pbzip2	c -Ipbzip2 -f	Parallel bzip2. pbzip2 process used about 900 MiB RAM at maximum.
lz4	0m3.914s	287M	lz4	c -I"lz4" -f	Really fast but the resulting archive is barely compressed. <i>Worst</i> compression king.
lz4	0m56.506s	207M	lz4 -12	c -I"lz4 -12" -f	Supports levels [1-12]. Uses 1 core, and there does not appear to be any multi-threaded variant.
lzip	4m42.017s	116M	lzip	c --lzip -f	v1.21. Standard lzip will only use <i>one</i> core (at 100%). Very slow.
lzip	0m42.542	118M	plzip	c -Iplzip -f	plzip 1.8 (Parallel lzip), default level (-6).
lzip	1m39.697s	110M	plzip -9	c -I"plzip -9" -f	Parallel lzip at best compression (-9). plzip process used 5.1 GiB RAM at maximum.
xz	5m2.952s	114M	xz	cfJ	Standard xz will only use one core (at 100%). Unbearably slow.

Algorithm	Time	Size	Command	Parameters	Comment
xz	0m53.569s	115M	pxz	c -Ipxz -f	Parallel PXZ 4.999.9beta. Process used 1.4 GiB RAM at maximum.
xz	1m33.441s	110M	pxz -9	c -I"pxz -9" -f	Parallel PXZ 4.999.9beta using its best possible compression. pxz process used 3.5 GiB at maximum.
zstd	0m3.034s	167M	zstd	c --zstd -f	zstd uses 1 core by default.
zstd	1m18.238s	117M	zstd -19 -T0	c -I"zstd -19 -T0" -f	-19 gives the best possible compression and -T0 utilizes all cores. If a non-zero number is specified, zstd uses that many cores.

A few minor points should be apparent from above numbers:

All the *standard* binaries GNU/Linux distributions give you as a *default* for all the commonly used compression algorithms are *extremely slow* compared to the parallel implementations that *are* available but *not* defaults.

This is true for bzip, there is a huge difference between 10 seconds and one minute. And it is specially true for [lzip](#) and xz, the difference between one minute and five is significant.

The difference between the `pigz` parallel implementation of `gzip` and regular `gzip` may *appear* to be small since both are *very fast*. The difference between 3 and 23 seconds is *huge* in terms of *percentage*.

[lzip](#) and xz offer the *best* compression. They are also the slowest alternatives. This is especially true if you do not use the parallel implementations.

Both plzip (5.1 GiB) and pxz (3.5 GiB at -9) use *a lot* of memory. Expect *much worse* performance on memory-constrained machines.

The difference between bzip2 and pbzip2 is *huge*. It may not appear that way since bzip is so much faster than xz and lzip but pbzip actually about *ten times faster* than regular bzip.

pbzip2's default compression is apparently it's best at -9. A close-up inspection of the output files reveal that they are identical (130260727b) with and without -9.

zstd, appears to be the clear winner, with leading compression speed, decompression speed, and acceptable compression ratio.

Algorithm	Time	Command	Parameters	Comments
none	0m1.204s	tar	-xf	Raw tar with no compression.
gzip2	0m4.232s	gzip2	-xfz	
gzip	0m2.729s	pigz	-x -Ipigz -f	gzip is a clear winner if decompression speed is the <i>only</i> consideration.
bzip2	0m20.181s	bzip2	xfj	
bzip2	0m19.533s	pbzip2	-x -Ipbzip2 -f	The difference between bzip2 and pbzip2 when <i>decompressing</i> is barely measurable
lzip	0m10.590s	lzip	-x --lzip -f	
lz4	0m1.873s	lz4	-x -Ilz4 -f	Fastest of them all but not very impressive considering the compression it offers is almost nonexistent.
lzip	0m8.982s	plzip	-x -Iplzip -f	
xz	0m7.419s	xz	-xfJ	xz offers the best decompression speeds of all the well-compressed algorithms.
xz	0m7.462s	pxz	-x -Ipxz -f	
zstd	0m3.095s	zstd	-x --zstd -f	When compressed with no options (the default compression level is 3).
zstd	0m2.556s	zstd	x --zstd -f	When compressed with <code>tar c -I"zstd -19 -T0"</code> (compression level 19)

Common Flag Usage

- `-v` → verbose, meaning print out status
- `-f` → file
- `-c` → create
- `-x` → extract
- `-k` → Don't replace existing files when extracting
- `-u` → Append files which are newer than the corresponding copy in the archive
- `-r` → Append files to the end of an archive
- `-l` → Print a message if not all links are dumped
- `-d` → find the difference between the archive and file system

- `-n` → assume the archive is seekable. This option takes effect only if the archive is open for reading
- `-t` → test/list content
- `-z` → filter the archive through **gzip**
- `--lzip` → filter the archive through **lzip**
- `--lzma` → filter the archive through **lzma**
- `--lzop` → filter the archive through **lzop**
- `-J` → filter the archive through **xz**

.tar - archiving utility (Default: no compression) - supports gzip and bzip2 compression

```
$ tar -cvf [file.tar] [files/directories] # create
$ tar -xvf [file.tar] # extract
$ tar -xvf [file.tar] -C [/path] # extract to a location
$ tar -xvf [file.tar] --overwrite # extract overwrite
$ tar -tvf [file.tar] # list content
```

Flag	Compression Method
<code>-z</code>	uses gzip
<code>-j</code>	uses bzip2

.gz - uses Lempel-Ziv coding (LZ77) for compression

```
$ gzip [files] # create .gz of each file (remove original files)
$ gzip -k [files] # create .gz of each file (keep original files)
$ gzip -r [directory] # create .gz of a directory files
$ gzip -d / gunzip [file.gz] # extract
$ gzip -dr / gunzip [directory] # extract a directory
$ gzip -dk [file.gz] # keep the original file in a directory
$ gzip -l [file.gz] # list content
$ gzip -df / gunzip -f [file.gz] # extract overwrite
```

.tar.gz - remember the flags given above as an alternative

```
$ tar -cvzf [file.tar.gz] [files] [directories] # create
$ tar -C [/path] -cvzf [file.tar.gz] [files_in_path] # create without including the path
$ tar -xvzf [file.tar.gz] # extract
$ tar -xvzf [file.tar.gz] -C [/path] # extract to a location
$ tar -xvzf [file.tar.gz] --overwrite # extract overwrite
$ tar -tvzf [file.tar.gz] # list content
```

.bz2 - uses Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by bzip command (LZ77/LZ78-based compressors)

```
$ bzip2 -z [files] # create .bz2 of each file (remove original files)
$ bzip2 -zk [files] # create gzip of each file (keep original files - no
directory support)
$ bzip2 -zd / bunzip2 [file.bz2] # extract
$ bzip2 -df / bunzip2 -f [file.bz2] # extract overwrite
```

.tar.bz2

```
tar -cvfj [archive.tar.bz2] [file1] [file2] [file3] # create an archive for
several files
tar -cvfj [archive.tar.bz2] [directory] # create an archive for a directory
tar -xvfj [archive_file.tar.bz2] [/path/to/file1] [/path/to/file2] # Extract
specific files from an archive
tar -xvfj [archive.tar.bz2] --wildcards '*.txt' # Extract files with extension
 '*.txt' from an archive
```

.XZ

```
$ xz -z [files] # create .xz of each file (remove original files)
$ xz -zk [files] # create .xz of each file (keep original files - no directory
support)
$ xz -zd /unxz [file.gz] # extract
$ xz -df /unxz [file.gz] # extract overwrite
```

.tar.xz

```
$ tar -cvJf [file.tar.gz] [files] [directories] # create
$ tar -xvJf [file.tar.gz] # extract
$ tar -xvJf [file.tar.gz] -C [/path] # extract to a location
$ tar -xvJf [file.tar.gz] --overwrite # extract overwrite
$ tar -tvJf [file.tar.gz] # list content
```

From Denny Zhang's Website

1.1 Decompress

Name	Summary
Unpack *.tar	<code>tar -xvf ./file.tar</code>
Unpack *.tar.gz	<code>tar xvzf ./file.tar.gz</code>
Unpack *.rar	<code>unrar e ./file.rar</code>
Unpack *.zip	<code>unzip ./file.zip</code>
Unpack *.gz	<code>gunzip ./file.gz; gzip -d file.gz</code>
Unpack *.bz2	<code>tar -jxvf file.tar.bz2 -C /tmp/extract_here/</code>
Unpack *.7z	<code>7z e ./file.7z</code> install p7zip first
Unpack *.xz	<code>tar -xf ./file.tar.xz</code>
Unpack *.jar	<code>jar -xvf ./file.jar</code>
Unpack *.war	<code>jar -xvf ./file.war</code>
Unpack *.tgz	<code>tar -xf ./file.tgz</code>

1.2 Compress Common Format

Name	Command
Pack to *.tar	<code>tar -cvf ./file.tar <dir_to_compress></code>
Pack to *.gz	<code>tar -zcvf ./newfile.tar.gz <dir_to_compress></code>
Pack to *.tgz	<code>tar -cvzf backup.tgz *</code>
Pack to *.bz2	<code>tar -cvf ./file.tar.bz2 <dir_to_compress></code>
Pack to *.zip	<code>zip -r ./file.zip <dir_to_compress></code>

1.3 Advanced Usage

Name	Command
Pack with exclude	<code>tar --exclude "*.git*" -cvf file.tar <dir_to_compress></code>
Remove a single file from tarball	<code>tar --delete -f file.tar file1.txt</code>
Add file to tarball. -r indicates append	<code>tar -rf file.tar file1.txt</code>
Create archive with password protection	<code>zip -er my.zip some_folder</code>

Pandoc - markup converter

```
$ curl cheat.sh/pandoc
```

```
# Convert documents between various formats.

# More information: <https://pandoc.org>.

# Convert file to pdf (the output format is determined by file extension):

pandoc input.md -o output.pdf

# Force conversion to use a specific format:

pandoc input.docx --to gfm -o output.md

# Convert to a standalone file with the appropriate headers/footers (for LaTeX,
HTML, etc.):

pandoc input.md -s -o output.tex

# List all supported input formats:

pandoc --list-input-formats

# List all supported output formats:

pandoc --list-output-formats
```

```
cory@amethyst:~$ pandoc --list-input-formats
commonmark
creole
csv
docbook
docx
dokuwiki
epub
fb2
gfm
haddock
html
ipynb
jats
jira
json
latex
man
markdown
markdown_github
markdown_mmd
markdown_phpextra
markdown_strict
mediawiki
muse
native
odt
```

```
opml
org
rst
t2t
textile
tikiwiki
twiki
vimwiki
```

```
cory@amethyst:~$ pandoc --list-output-formats
```

```
asciidoc
asciidocctor
beamer
commonmark
context
docbook
docbook4
docbook5
docx
dokuwiki
dzslides
epub
epub2
epub3
fb2
gfm
haddock
html
html4
html5
icml
ipynb
jats
jats_archiving
jats_articleauthoring
jats_publishing
jira
json
latex
man
markdown
markdown_github
markdown_mmd
markdown_phpextra
markdown_strict
mediawiki
ms
muse
native
odt
opendocument
opml
org
pdf
plain
pptx
revealjs
```



```
rst
rtf
s5
slideous
slidy
tei
texinfo
textile
xwiki
zimwiki
```

Scanning

Setup a Fujitsu ix500 scanner on Linux

Source: https://ridaayed.github.io/posts/setup_fujitsu_ix500_scanner_linux/

Install package

```
sudo apt install sane-utils -y
```

Approve the connection

Enable the scanner by opening the scanner's lid

```
sudo sane-find-scanner -q
```

```
found USB scanner (vendor=0x04c5 [Fujitsu], product=0x132b [ScanSnap ix500]) at
libusb:004:002
found USB scanner (vendor=0x0bda [Generic], product=0x0139 [USB2.0-CRW]) at
libusb:003:004
```

Get a list of devices:

```
scanimage --list-devices
```

```
device `fujitsu:ScanSnap ix500:15419' is a FUJITSU ScanSnap ix500 scanner
```

Get a list of all device specific options:

```
scanimage --device 'fujitsu:ScanSnap ix500:15419' --all-options
```

Output format is not set, using pnm as a default.

All options specific to device `fujitsu:ScanSnap ix500:15419':

Standard:

```
--source ADF Front|ADF Back|ADF Duplex [ADF Front]
    Selects the scan source (such as a document-feeder).
--mode Lineart|Gray|Color [Lineart]
    Selects the scan mode (e.g., lineart, monochrome, or color).
--resolution 50..600dpi (in steps of 1) [600]
```

Sets the resolution of the scanned image.

Geometry:

- page-width 0..221.121mm (in steps of 0.0211639) [215.872]
Specifies the width of the media. Required for automatic centering of sheet-fed scans.
- page-height 0..876.695mm (in steps of 0.0211639) [279.364]
Specifies the height of the media.
- l 0..215.872mm (in steps of 0.0211639) [0]
Top-left x position of scan area.
- t 0..279.364mm (in steps of 0.0211639) [0]
Top-left y position of scan area.
- x 0..215.872mm (in steps of 0.0211639) [215.872]
Width of scan-area.
- y 0..279.364mm (in steps of 0.0211639) [279.364]
Height of scan-area.

Enhancement:

- brightness -127..127 (in steps of 1) [0]
Controls the brightness of the acquired image.
- contrast -127..127 (in steps of 1) [0]
Controls the contrast of the acquired image.
- threshold 0..255 (in steps of 1) [0]
Select minimum-brightness to get a white point
- rif[=(yes|no)] [no]
Reverse image format
- ht-type Default|Dither|Diffusion [inactive]
Control type of halftone filter
- ht-pattern 0..3 (in steps of 1) [inactive]
Control pattern of halftone filter
- emphasis -128..127 (in steps of 1) [0]
Negative to smooth or positive to sharpen image
- variance 0..255 (in steps of 1) [0]
Set SDTC variance rate (sensitivity), 0 equals 127

Advanced:

- ald[=(yes|no)] [no]
Scanner detects paper lower edge. May confuse some frontends.
- df-action Default|Continue|Stop [Default]
Action following double feed error
- df-skew[=(yes|no)] [inactive]
Enable double feed error due to skew
- df-thickness[=(yes|no)] [inactive]
Enable double feed error due to paper thickness
- df-length[=(yes|no)] [inactive]
Enable double feed error due to paper length
- df-diff Default|10mm|15mm|20mm [inactive]
Difference in page length to trigger double feed error
- bgcolor Default|White|Black [Default]
Set color of background for scans. May conflict with overscan option
- dropoutcolor Default|Red|Green|Blue [Default]
One-pass scanners use only one color during gray or binary scanning, useful for colored paper or ink
- buffermode Default|Off|On [Off]
Request scanner to read pages quickly from ADF into internal memory
- prepick Default|Off|On [Default]
Request scanner to grab next page from ADF
- overscan Default|Off|On [Default]
Collect a few mm of background on top side of scan, before paper enters ADF, and increase maximum scan area beyond paper size, to allow collection on remaining sides. May conflict with bgcolor option

```

--sleeptimer 0..60 (in steps of 1) [0]
    Time in minutes until the internal power supply switches to sleep mode
--offtimer 0..960 (in steps of 1) [240]
    Time in minutes until the internal power supply switches the scanner
    off. Will be rounded to nearest 15 minutes. Zero means never power off.
--lowmemory[=(yes|no)] [no]
    Limit driver memory usage for use in embedded systems. Causes some
    duplex transfers to alternate sides on each call to sane_read. Value of
    option 'side' can be used to determine correct image. This option
    should only be used with custom front-end software.
--side[=(yes|no)] [no] [read-only]
    Tells which side (0=front, 1=back) of a duplex scan the next call to
    sane_read will return.
--swdeskew[=(yes|no)] [no]
    Request driver to rotate skewed pages digitally.
--swdespeck 0..9 (in steps of 1) [0]
    Maximum diameter of lone dots to remove from scan.
--swcrop[=(yes|no)] [no]
    Request driver to remove border from pages digitally.
--swskip 0..100% (in steps of 0.100006) [0]
    Request driver to discard pages with low percentage of dark pixels

```

Sensors:

```

--top-edge[=(yes|no)] [no] [hardware]
    Paper is pulled partly into ADF
--a3-paper[=(yes|no)] [no] [hardware]
    A3 paper detected
--b4-paper[=(yes|no)] [no] [hardware]
    B4 paper detected
--a4-paper[=(yes|no)] [no] [hardware]
    A4 paper detected
--b5-paper[=(yes|no)] [no] [hardware]
    B5 paper detected
--page-loaded[=(yes|no)] [no] [hardware]
    Page loaded
--omr-df[=(yes|no)] [no] [hardware]
    OMR or double feed detected
--cover-open[=(yes|no)] [no] [hardware]
    Cover open
--power-save[=(yes|no)] [no] [hardware]
    Scanner in power saving mode
--email[=(yes|no)] [no] [hardware]
    Email button
--manual-feed[=(yes|no)] [no] [hardware]
    Manual feed selected
--scan[=(yes|no)] [no] [hardware]
    Scan button
--function <int> [1] [hardware]
    Function character on screen
--double-feed[=(yes|no)] [no] [hardware]
    Double feed detected
--error-code <int> [0] [hardware]
    Hardware error code
--skew-angle <int> [0] [hardware]
    Requires black background for scanning

```

Scanning multiple pages (where each page is a separate file) as colored .PNG image files

```
scanimage -y 297 --format png --batch=scan%04d.png --batch-count=10 --resolution 300 --mode Color
```

Installing the package img2pdf in order to convert .PNGs or .JPEGs to PDF files

```
sudo apt install img2pdf
```

Verifying the package version

```
img2pdf -V
```

Converting and combining scanned .PNGs into a single PDF file

```
img2pdf *.png -o pcmag_IT_certification_article.pdf
```

Using both commands in a "one-liner" command (The ; indicates that the commands should run sequentially as separate commands)

```
scanimage -y 297 --format png --batch=scan%04d.png --batch-count=10 --resolution 300 --mode Color; img2pdf *.png -o pcmag_IT_certification_article.pdf
```

Note: I know what the `--batch` option does: it allows for multiple pages to be scanned with an Automatic Document Feeder (ADF) scanner. However, when I tried to change the `scan%04d.png` filename, this would result in an error. Given my intentions (as explained above), the name of the scanned .PNG files is irrelevant, because they are combined into a single PDF file anyway. Each time you scan something with the scanner, create a directory within the `~/scans` directory for your scanned image files (.PNGs or .JPEGs).

1. Now, to

convert all images to pdf in one-go

```
img2pdf <input_file> -o <output_pdf>
```

For instance,

```
img2pdf *.png -o outcome.pdf
```

2. Or, to

convert only one image file to pdf

-

```
img2pdf A.png -o outcome.pdf
```

3. Specify the files to be converted to pdf

-

```
img2pdf B.png D.png E.png -o outcome.pdf
```

In this case only B.png, D.png and E.png will be converted to outcome.pdf

4. Up till now, you may have noticed that images are converted inline with their respective sizes. We can also

specify page size

as well -

```
img2pdf B.png D.png E.png --pagesize 18cmx12cm -o outcome.pdf
```

5. Or, to

specify image size

-

```
img2pdf B.png D.png E.png --imgsize 18cmx12cm -o outcome.pdf
```

#

Cryptography

Encryption - application of cryptography

confidentiality - secrecy

cipher - encryption algorithm

key - secret value required to decrypt

Symmetric encryption - the key used to decrypt is the same as the key used to encrypt

Asymmetric encryption (aka public-key encryption) - the key used to decrypt is different from the key used to encrypt

Cleartext - information that is stored or sent in an unencrypted form. It is already in its expected form, consumable and readable. Cleartext has not been subject to encryption whatsoever, and there is no expectation that it has been.

Plaintext - specifically refers to information that is inputted into a cipher, or encryption algorithm.

Ciphertext is information that is unreadable once it has passed through a cipher or encryption algorithm.

Storing data in cleartext is an invitation to its theft, altering, destruction, unauthorized transmission, unsanctioned disclosure, and the like. The database or system where cleartext passwords are stored, for example, are often protected with passwords and other shared secrets such as one-time passwords (OTPs).

Encryption turns a plaintext into a ciphertext.

Decryption turns a ciphertext back into a plaintext.

Encryption

$$C = E(K, P)$$

C represents a ciphertext

E represents a cipher algorithm (encryption mode)

K represents a key

P represents a plaintext

Decryption

$$P = D(K, C)$$

D represents a cipher algorithm (decryption mode)

For some ciphers, the ciphertext is the same size as the plaintext; for some others, the ciphertext is slightly longer. However, ciphertexts can never be shorter than plaintexts.

The Caesar cipher is so named because the Roman historian Suetonius reported that Julius Caesar used it. It encrypts a message by shifting each of the letters down three positions in the alphabet, wrapping back around to A if the shift reaches Z.

The Caesar cipher is super easy to break: to decrypt a given ciphertext, simply shift the letters three positions back to retrieve the plaintext. That said, the Caesar cipher may have been strong enough during the time of Crassus and Cicero. Because no secret key is involved (it's always 3), users of Caesar's cipher only had to assume that attackers were illiterate or too uneducated to figure it out—an assumption that's much less realistic today.

It took about 1500 years to see a meaningful improvement of the Caesar cipher in the form of the Vigenère cipher, created in the 16th century by an Italian named Giovan Battista Bellaso. The name "Vigenère" comes from the Frenchman Blaise de Vigenère, who invented a different cipher in the 16th century, but due to historical misattribution, Vigenère's name stuck. Nevertheless, the Vigenère cipher became popular and was later used during the American Civil War by Confederate forces and during WWI by the Swiss Army, among others.

The Vigenère cipher is similar to the Caesar cipher, except that letters aren't shifted by three places but rather by values defined by a key, a collection of letters that represent numbers based on their position in the alphabet. For example, if the key is DUH, letters in the plaintext are shifted using the values 3, 20, 7 because D is three letters after A, U is 20 letters after A, and H is seven letters after A. The 3, 20, 7 pattern repeats until you've encrypted the entire plaintext. For example, the word CRYPTO would encrypt to FLFSNV using DUH as the key: C is shifted three positions to F, R is shifted 20 positions to L, and so on. Figure 1-3 illustrates this principle when encrypting the sentence THEY DRINK THE TEA.

The Vigenère cipher is clearly more secure than the Caesar cipher, yet it's still fairly easy to break. The first step to breaking it is to figure out the key's length. For example, take the example in Figure 1-3, wherein THEY DRINK THE TEA encrypts to WBLBXYLHRWBLWYH with the key DUH. (Spaces are usually removed to hide word boundaries.) Notice that in the ciphertext WBLBXYLHRWBLWYH, the group of three letters WBL appears twice in the ciphertext at nine-letter intervals. This suggests that the same three-letter word was encrypted using the same shift values, producing WBL each time. A cryptanalyst can then deduce that the key's length is either nine or a value divisible by nine (that is, three). Furthermore, they may guess that this repeated three-letter word is THE and therefore determine DUH as a possible encryption key.

The second step to breaking the Vigenère cipher is to determine the actual key using a method called frequency analysis, which exploits the uneven distribution of letters in languages. For example, in English, E is the most common letter, so if you find that X is the most common letter in a ciphertext, then the most likely plaintext value at this position is E.

GNU Privacy Guard

Better Living through GPG

On Linux, the Swiss Army knife of encryption is **Werner Koch's Gnu Privacy Guard (GnuPG)** suite. Its core applications command (`gpg`) conjures memories of **Pretty Good Privacy (PGP)**, a crypto tool originally released in 1991 by **Phil Zimmermann**. Nowadays, OpenPGP is a standard specifying how encrypted messages and the bits associated with them should be stored and GnuPG fully implements this standard. GnuPG works from the command line and has a reputation of being complicated and unfriendly (see <https://moxie.org/blog/gpg-and-me>). It avails the user of all the modern private and public key algorithms, as well as all manner of other knobs to twiddle. As a result, there are a huge number of command-line options and the man pages make for some lengthy reading. Most distros will install GnuPG as standard, so we shouldn't have to install anything for this tutorial.

Traditional, symmetric encryption (where two parties share a secret key or password) is all very well, but it relies on the communicating parties

Generate Public and Private Keys

First, generate your public and private key pair.

```
gpg --gen-key
```

Just follow the interactive command line instruction.

After it's done, your key will be at the dir `~/.gnupg`. Note: the key files there are all binary files.

Be sure you remember your passphrase.

here's sample session:

```
xah@xah-p6813w ◆ 2014-02-22 19:40 ◆ ~
◆ gpg --gen-key
gpg (GnuPG) 1.4.11; Copyright (C) 2010 Free Software Foundation, Inc.
```

This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory `/home/xah/.gnupg' created
gpg: new configuration file `/home/xah/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/xah/.gnupg/gpg.conf' are not yet active during
this run

gpg: keyring `/home/xah/.gnupg/secring.gpg' created
gpg: keyring `/home/xah/.gnupg/pubring.gpg' created

Please select what kind of key you want:

- (1) RSA and RSA (default)
- (2) DSA and ElGamal
- (3) DSA (sign only)
- (4) RSA (sign only)

Your selection?

RSA keys may be between 1024 and 4096 bits long.

What keysize do you want? (2048)

Requested keysize is 2048 bits

Please specify how long the key should be valid.

- 0 = key does not expire
- <n> = key expires in n days
- <n>w = key expires in n weeks
- <n>m = key expires in n months
- <n>y = key expires in n years

Key is valid for? (0) 1y

Key expires at Sun 22 Feb 2015 09:02:45 PM PST

Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:

"Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Xah Lee

Email address: xah@xahlee.org

Comment:

You selected this USER-ID:

"Xah Lee <xah@xahlee.org>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o

You need a Passphrase to protect your secret key.

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

Not enough random bytes available. Please do some other work to give
the OS a chance to collect more entropy! (Need 282 more bytes)

...+++++

.....+++++

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

Not enough random bytes available. Please do some other work to give
the OS a chance to collect more entropy! (Need 76 more bytes)


```
+++++
```

```
Not enough random bytes available. Please do some other work to give
the OS a chance to collect more entropy! (Need 28 more bytes)
```

```
.+++++
```

```
gpg: /home/xah/.gnupg/trustdb.gpg: trustdb created
gpg: key 07438185 marked as ultimately trusted
public and secret key created and signed.
```

```
gpg: checking the trustdb
```

```
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
```

```
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
```

```
gpg: next trustdb check due at 2015-02-23
```

```
pub 2048R/07438185 2014-02-23 [expires: 2015-02-23]
```

```
Key fingerprint = C3B1 AFF5 C285 F48E 3FB5 EA61 C811 6AFD 0743 8185
```

```
uid Xah Lee <xah@xahlee.org>
```

```
sub 2048R/3668BBB3 2014-02-23 [expires: 2015-02-23]
```

Don't worry if you made a mistake. If this is your first time using gpg, you can delete the whole `~/.gnupg` dir and try again.

{List, Import/Add, Export, Find}, Keys

List Keys

- `gpg --list-keys`

Lists all public keys.

Here's sample output:

```
◆ gpg --list-key
/home/xah/.gnupg/pubring.gpg
-----
pub 2048R/07438185 2014-02-23 [expires: 2015-02-23]
uid Xah Lee <xah@xahlee.org>
sub 2048R/3668BBB3 2014-02-23 [expires: 2015-02-23]

pub 2048R/5BB6FE3C 2014-02-26 [expires: 2014-05-27]
uid Xah Lee <xahlee@gmail.com>
sub 2048R/779449FC 2014-02-26 [expires: 2014-05-27]
```

Note: in the line `pub 2048R/07438185 ...`, the number after the slash is the **key ID**. Key ID is useful in many commands.

- `gpg --list-sigs`

List all public keys and signatures.

Export Your Public Keys

now, export your public key.

- `gpg --export -a`

Prints your public key as plain text to screen.

- `gpg --local-user key_id --export -a`

Prints a specific public key as plain text.

You can place your public key on your website, blog, send it to friend, etc. People need your public key to send you encrypted files or verify your signature.

Note: somebody else can generate a key pair and send to your friend Alice by email and forge email header to pretend it's from you. And your friend might be fooled to encrypted a big secret using this guy's public key thinking it's yours. In general, you shouldn't trust public keys easily. The detailed security risk is beyond this simple tutorial.

Publish Your Key to Public Key Server

You can send your key to a public-key server, so that other people can easily find your key in order to send encrypted messages to you.

- `gpg --send-keys key_id`

Upload your public key to default key server.

sample:

```
◆ gpg --send-keys 5BB6FE3C
gpg: sending key 5BB6FE3C to hkp server keys.gnupg.net
```

You can find your key_id by `gpg --list-keys`, it's the number after the slash.

the default key server is `hkp://keys.gnupg.net/`

Import/Add Other People's Public Keys

To send a encrypted message to Alice, you need her public key.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.11 (GNU/Linux)

mQENBFMJgWoBCADTLgXzbrSyPazY+M8j7trJ9hk4B1WYMd79ut7/fEq39s7iR3eb
7LH8ngbnlLsweNwaLZl3LGP3Yu1sCW9lB2nMw8BaP9B5/NN+YaX4y3XJiz10p3JO
jW4cz12Pd/kOrUyA0ViNJStPYVmX0JZ0BvwZzxXJPuiJRbWwWZY5VKhjdwEUycF3
GpBKLB0X8uGXXhFIeKQ2v4mZIBYj8vdKbGIPBDF0tRLbhY/sj2je6Er8XF7BIFBQ
msEnGj0kehxqCv6RZkd8RdHTGItOzG+qowitG6YG/rjFY3Mrranc41pAB4tZ0IKU
oX+/rZR0oiiZcRnL9SRvk2sHP/D/FG68QiR/ABEBAAG0GFhhaCBMZUgPHhhaEB4
YWhsZWUub3JnPokBPgQTAQIAKAUCUwmBagIbAwUJAeEzgaYLCQgHAWIGFQgCCQoL
BBYCAwECHgECF4AACgkQyBFq/QdDgYV72wgA0spK+mLZv5RGym9BNleDfi531sF3
I4KrI4t4SuBNA1u5l7wvCm7+SWN/oiuZjkhIXd30MeePmH3D9JnopLbMQGtYYR4W
q0YpGbkoE898gH/Bm3+IiRkQthCAjxWKEGkRLqeQq2Td4/TxTT9oDg+eHZNtcEsp
0WU+/JEeaFDiAIGp1c0Z5PyZ1g/upnECZ3HD/gkFUTbBeNjMJCW1HT7l2yBa/SMT
8lcpdWACKDjqik6hvo3fZMILlyQg+5zbGLPhrkn9R920CK8Ewc4zJVjwWslxqy5
Dsjs1Lh0SXfb2TEgkGvvp1Pn8ni0VimhytY+zEyMm/qz17aCbSSt5bi3cLkBDQRT
CYFqAQgAxy3iLFGv583IhDXphiq04f92BunbNCdwLvdDqzPuFdp+YfwEkvyTBKS
PiFrNWXxiTThoIbSQSMEkQnyxZkWTIK2UBe1Qpb7B+oSPRLSMVCN+CTc87FiEJnT
iY5x7gtzeCKRUguMXfY1WxyI0gsrichRF8Yy/wQgGgXCpyzYDlcYxMtGxMl0Nq/
5kma9T64EmT62j1sPvVbqwnRbiMdYld5I31tISI1UZMPvF9wkLk9F92TDbmJzR4c
0Xx9tGsa2FYK/vmh5TruA0bi+Zs0fjeRhB6xFXouE/bDM0V8pCCj32yjwesBeM7J
6nVJIF5nf/+epiur9E5DvpYA1HR6TWARAQABiQELBBgBAGAPBQJTCYFqAhsMBQkB
4TOAAAOJEMgRav0HQ4GFVQsH/2Xip8v8Ux8zSoGbKmdnF39Ma8cOqEw6RlOfkBRu
voXsedZwr5NpAndhrM1oMIPeZU8tXLKzhJLYENqiJ/FTWZVmqKQDcqI40ymvT4C4
Cu/3luqGC0iPpy+qmg5nQzi14205SXwM8hQ9/8RZKuI2gtFA8eP7G8mhsIPG73a
UGaquh1p3fxa8MMjxPXd6y40PM71NSQ9rkbwm9Zx7MU3cp0jrXXxtM52WnaxtaIR
```

```
Mycv7Axvv9MRDgTI5ozmxPxbothLKyk5XFCG6Tv8zro911qW6vFCfzpQqu01HTtFM
l9xhK+JJFC/S02oSALpBG5noAoxmM4VILvjMzIU7nS65Quw=
=PwPW
-----END PGP PUBLIC KEY BLOCK-----
```

Find People's Keys

Often, people's keys are on a key server.

- `gpg --search-keys name`

Search a key by name. The name is any string, usually people's name or email address.

sample output:

```
◆ gpg --search-keys xah
gpg: searching for "xah" from hkp server keys.gnupg.net
(1)   Xah Lee <xahlee@gmail.com>
      2048 bit RSA key 5BB6FE3C, created: 2014-02-26
(2)   Xah Lee <xah@xahlee.org>
      2048 bit RSA key 07438185, created: 2014-02-23
(3)   Łukasz Rumiński <xah-luk@o2.pl>
      2048 bit RSA key CA9CBC6B, created: 2012-04-12
(4)   Łukasz Rumiński <xah-luk@o2.pl>
      2048 bit RSA key C26152F8, created: 2012-03-29
(5)   Xah Lee <xah@xahlee.org>
      1024 bit DSA key CB8F3E74, created: 2000-03-25
Keys 1-5 of 5 for "xah".  Enter number(s), N)ext, or Q)uit > q
```

Once you find the person, you can import the key by:

- `gpg --recv-keys key_id`

Import a key identified by `key_id` to your keyring.

Revoke Key

if your computer got stolen (your secret is stolen), you need to revoke your key.

to revoke, you need your passphrase, and your secret key in your computer.

- `gpg --gen-revoke`

Revoke your key.

Sign Documents

you can use `gpg` to digitally sign documents, so that others know you signed it.

- `gpg --sign file`

Create a signed file. This will create a file in the same path, with `.gpg` suffix in file name. The created file is compressed binary file.

- `gpg --clearsign file`

Same as `--sign`, but create a plain text file. The newly created file will have `.asc` name suffix.

Sign Documents with Different Key

gpg lets you create many key pairs. To specify a particular key, use the `--local-user` option, like this:

- `gpg --local-user xahlee@gmail.com --sign myfile.txt`

The `--local-user` option can be used with other commands to specify a key.

Verify Signed Documents

```
gpg --verify file
```

You need the other person's public key in your keyring. If you haven't done so, use `gpg --import key_file_name`.

Encrypt File

you can encrypt your files.

- `gpg --encrypt file_name`

Encrypt a file using your own public key. So, only yourself can decrypt it later.

- `gpg --encrypt --recipient name file_name`

Encrypt a file for name to read, using name's public key. The name is the "uid" in `man gpg doc`.

You must have name's public key in your keyring. If not, use `--import` first.

`gpg --list-public-keys` to show a list of name (uid).

When you encrypt a file for Alice to read, you can and should also sign the file. So, when Alice reads it, she also has confidence that YOU encrypted it, not somebody else.

- `gpg --encrypt --recipient name --sign file_name`

Encrypt a file for name to read, and sign it with your name.

Decrypt File

- `gpg --decrypt file_name`

Decrypt a file using your secret key. The file must be one that's encrypted by your public key. The output is printed to the screen.

- `gpg --decrypt file_name > output_file_name`

Decrypt a file and save it as `output_file_name`.

Cybersecurity Tools for configuring and troubleshooting network connectivity

Linux and other Unix-based systems often offer several alternatives for solving a single problem. Although this approach can be confusing to some novice administrators, it is one of the defining features of the open source environment.

The networking tool collection is no exception to this practice. You'll find an array of useful tools - some overlapping and some unique - for configuring, managing, and troubleshooting network connections.

In this article, we highlight some favorite tools in the networking collection. Of course, a full description of the complete TCP/IP networking environment could fill up a very long book. Here are some basics related to the TCP/IP and OSI models.

These two acrostics go from physical layer to application layer:

- Please Do Not Throw Sausage Pizza Away
- Please Do Not Touch Steve's Pet Alligator

And these acrostics go the other direction, from application to physical:

- All People Seem To Need Data Processing (a popular one)
- All People Standing Totally Naked Don't Perspire (hmm, that's interesting!)
- All People Should Teach Networking Daily Please

[OSI Model](#) by [GhaziAnwar](#)

Table 7.2: OSI Reference Model Layer Summary

#	Layer Name	Key Responsibilities	Data Type Handled	Scope	Common Protocols and Technologies
1	Physical	Encoding and signaling; physical data transmission; hardware specifications; topology and design	Bits	Electrical or light signals sent between local devices	Physical layers of most of the technologies listed for the data link layer
2	Data Link	Logical link control; media access control; data framing; addressing; error detection and handling; defining requirements of physical layer	Frames	Low-level data messages between local devices	IEEE 802.2 LLC, Ethernet family; Token Ring; FDDI and CDDI; IEEE 802.11 (WLAN, Wi-Fi); HomePNA; HomeRF; ATM; SLIP and PPP
3	Network	Logical addressing; routing; datagram encapsulation; fragmentation and reassembly; error handling and diagnostics	Datagrams/packets	Messages between local or remote devices	IP; IPv6; IP NAT; IPsec; Mobile IP; ICMP; IPX; DLC; PLP; routing protocols such as RIP and BGP
4	Transport	Process-level addressing; multiplexing/demultiplexing; connections; segmentation and reassembly; acknowledgments and retransmissions; flow control	Datagrams/segments	Communication between software processes	TCP and UDP; SPX; NetBEUI/NBF
5	Session	Session establishment, management, and termination	Sessions	Sessions between local or remote devices	NetBIOS, Sockets, named pipes, RPC
6	Presentation	Data translation; compression and encryption	Encoded user data	Application data representations	SSL; shells and redirectors; MIME
7	Application	User application services	User data	Application data	DNS; NFS; BOOTP; DHCP; SNMP; RMON; FTP; TFTP; SMTP; POP3; IMAP; NNTP; HTTP; Telnet

The *Network Interface layer* of the TCP/IP model is equivalent to the *data link layer* (layer 2) in the OSI Reference Model and is sometimes called the *link layer*.

LINK LAYER Services

- *Framing, link access*
 - Encapsulates datagram into frame, adding the header and trailer
 - Channel access if shared medium
 - "MAC" addresses used in frame headers to identify the source and destination
 - Different from IP address!
- *Reliable delivery between adjacent nodes*
 - We learned how to do this already (chapter 3)!
 - Seldom used on low-bit error link (fiber, some twisted pair)
 - Wireless links: high error rates
 - Q: Why both link-level and end-end reliability?
- *Flow control*
 - Pacing between adjacent sending and receiving nodes
- *Error detection*
 - Errors caused by signal attenuation, noise.
 - Receiver detects presence of errors:
 - Signals sender for retransmission or drops frame
- *Error correction*
 - Receiver identifies and corrects bit error(s) without resorting to retransmission
- *Half-duplex and full-duplex*
 - With half-duplex, nodes at both ends of link can transmit, but not at the same time

LINK LAYER Location

- In each and every host
- Link layer implemented in "adapter" (aka *network interface card (NIC)*) or on a chip
 - Ethernet card, 802.11 card; Ethernet chipset
 - Implements the link layer and the physical layer
- Attached to host system buses
- Combination of hardware, software, and firmware

On the sending side, the controller takes a datagram that has been created and stored in host memory by the higher layers of the protocol stack, encapsulates the datagram in a link-layer frame (filling in the frame's various fields), and then transmits the frame into the communication link, following the link-access protocol. On the receiving side, a controller receives the entire frame, and extracts the network-layer datagram. If the link layer performs error detection, then it is the sending controller that sets the error-detection bits in the frame header and it is the receiving controller that performs error detection.

Figure 5.2 shows a network adapter attaching to a host's bus (e.g., a PCI or PCI-X bus), where it looks much like any other I/O device to the other host components. Figure 5.2 also shows that while most of the link layer is implemented in hardware, part of the link layer is implemented in software that runs on the host's CPU. The software components of the link layer implement higher-level link-layer functionality such as assembling link-layer addressing information and activating the controller hardware. On the receiving side, link-layer software responds

to controller interrupts (e.g., due to the receipt of one or more frames), handling error conditions and passing a datagram up to the network layer. Thus, the link layer is a combination of hardware and software—the place in the protocol stack where software meets hardware. Intel [2012] provides a readable overview (as well as a detailed description) of the 8254x controller from a software-programming point of view.

Interfaces

The name starts with two characters for the type of interface: `en` for Ethernet, `wl` for WLAN, `ww` for WWAN. The type is then distinguished: `o<Index>` represents an onboard interface with an ordinal number; `s<Slot>` is a slot card with an ordinal number, and `p<Bus>s<Slot>` reveals the location of the PCI card. Examples of the results are, say, `enp2s0` or `enp2s1`. This shows just a small part of the naming scheme; more information can be found online [4] [5]. I like the documentation at [5].

Version 197 of the `systemd` startup daemon unveiled a new method for naming devices. Instead of assigning consecutive device numbers to network devices, `systemd` assigns a *predictable* network device name based on identifying information about the device itself, such as:

- information provided in the BIOS
- the physical location of the hardware
- the interface's MAC (hardware) address

The system uses this information to assign a unique (and reproducible) number for the device. This number is then combined with a two-character prefix, such as `en` for wired Ethernet or `wl` for wireless LAN. For instance, an Ethernet adapter might have a logical name like `enp0s31f6` and a wireless network interface might have the logical name `wlp4s0`. The examples in this article use the interface name `enp0s31f6` - if you try these commands, change `enp0s31f6` to the logical name of your own network adapter.

The `ifconfig` command was, and still is on many systems, the default tool for configuring network interfaces. However, `ifconfig` is often considered obsolete, in that newer tools are provided for systems running kernels newer than 2.0. The `ifconfig` command is still available as part of the `net-tools` package, though, in all likelihood, it is automatically installed on your system.

On newer Linux systems, you also get the `ip` command. More than just a newer version of `ifconfig`, `ip` is the workhorse of the new generation of network tools. Not only does it integrate the functionality of several older tools, but `ip` also provides a unified syntax across all the various functions. In contrast, the utilities provided by the `net-tools` package are a patchwork collection of tools that were developed individually over many years.

ip and ifconfig

`ip` commands are typically more complex than their `ifconfig` equivalents. For instance, to display the configuration of the `enp0s31f6` interface, the `ifconfig` command is simply `ifconfig enp0s31f6`.

The `ip` command is part of the `iproute` package. The similarity between the tools in this package enables you to master the configuration of your network more quickly because you do not need to learn different syntax options for different functions. Furthermore, you don't need to remember which utility does what because, for the most part, `ip` integrates the capabilities of `ifconfig`, `route`, and `arp` into a single tool.

The generic usage is

```
ip [OPTIONS] <OBJECT> [COMMAND]
```

where OBJECT is something like *ip* for your IP configuration, *link* for a network interface, *addr* for your IP address, *route* for routes, and so forth. (The `ip` command also supports several other objects - see the `ip` man page for more details).

In the context of the `ip` command, a "link" is a network device, real or virtual. To display the details of a specific interface, you might enter the following:

```
ip addr show dev enp0s31f6
```

This command might give you something like the output shown below.

```
jccasad@Thinkie:~$ ip addr show dev enp0s31f6
2: enp0s31f6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast sta
link/ether 54:ee:75:a4:ea:b3 brd ff:ff:ff:ff:ff:ff
inet 10.0.0.121/24 brd 10.0.0.255 scope global dynamic enp0s31f6
    valid_lft 2415sec preferred_lft 2415sec
inet6 fe80::2cf8:b850:3274:fcde/64 scope link
    valid_lft forever preferred_lft forever
jccasad@Thinkie:~$
```

In most cases, the default argument is *show*, which displays the basic parameters of the given object. The default behavior is to display the information for all objects if none is specified. For example, `ip addr` will show (i.e., display) the address information about all network interfaces. If you want, you can use `list` instead of `show`. (Most users think about "listing" devices rather than "showing" them.)

This form of the `ip addr` command is composed of three parts: `show dev enp0s31f6`. One could say that the command-within-a-command is *show* with *dev enp0s31f6* acting as arguments.

If you want to add a virtual interface called `enp0s31f6:1`, the command would look like this:

```
ip addr add 192.168.1.42 dev enp0s31f6:1
```

In this case, you can think of `192.168.1.42 dev enp0s31f6:1` as arguments to the *add* command. The example here adds the IP address `192.168.1.42` to the device `enp0s31f6:1`.

With the `ip` command, you can also enable and disable interfaces (i.e., bring them up or down):

```
ip link set up dev enp0s31f6
```

In this example, the command *set*; *set* and *view* are the two options the *link* object accepts.

The `ifup` command is another option for starting up a network interface. As you would expect, there is also an `ifdown` command, which is a symbolic link to `ifup`.

Routing

If you were to imagine that network traffic is something like a letter you're sending to a friend, you would know their name (hostname) and address (IP address), and all you need to do to get the letter to your friend is address the letter, put a stamp on it, and stick it in the mailbox (gateway). However, between your gateway and the friend's gateway, the letter has to be *routed* between you and your friend.

The post office maintains a set of routes to deliver letters, and hosts on the Internet maintain routing tables to send packets to and fro. Usually the only route you have to worry about is the *default gateway* for your system. On your home LAN, this is the modem or DSL router that connects you to the Internet. For example, on my home network, the default route is 192.168.1.254.

The need to define network routes manually has decreased over the years. Most home and office networks today provide dynamic IP address assignment through DHCP, which includes information on the default gateway for the network.

Even if you explicitly define your network configuration - at installation or through the resident configuration utility for your Linux distribution - the tool typically asks you to specify a default gateway and the rest of the routing happens automatically. However, for complex network configurations, such as computers with multiple network interfaces or routed networks with multiple paths, you might occasionally have the need to add information directly to the routing table - for configuration, optimization, or troubleshooting purposes. If your system loses or doesn't have a default route and it isn't a gateway, then you're not going to be delivering any packets.

The traditional method for adding and managing routes is the `route` command. The `ip route` command is a more recent alternative with similar functionality.

Adding a route with `route` will look something like this:

```
route add -net 192.168.42.0/24 gw 192.168.1.99
```

The same thing with the `ip` command would look like:

```
ip route add 192.168.42.0/24 via 192.168.1.99
```

As you can see, the format is basically the same as when you added IP addresses. In this case, the object is a route and the command is *add*. Note that both commands add the route for a range of IP addresses (*192.168.42.0/24* - in CIDR format), and this route is assigned to a router address - with the *gw* ("gateway") argument in the *route* command and the more intuitive *via* with *ip route*.

If you enter the `ip route` command without any modifying arguments, you are shown the list of configured routes. Although this is not any simpler than running *route*, I think the output is a little more useful. For example, the output for the default route shown by *ip* is:

```
default via 192.168.2.1 dev enp0s31f6
```

whereas the output for *route* is more elaborate (Figure 2).

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.2.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
169.254.0.0	0.0.0.0	255.255.0.0	U	0	0	0	eth0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	192.168.2.1	0.0.0.0	UG	0	0	0	eth0

With the use of *route* and *ip*, you can hand-configure routes aside from the gateway. Say you have two interfaces on want to ensure that the *enp0s31f6* interface is used for the *192.168.42.0/24* network:

```
route add -net 192.168.42.0 netmask 255.255.255.0 gw 192.168.1.254 dev enp0s31f6
```

Now traffic headed to *192.168.42.0* will go through *enp0s31f6*. One caveat - this all goes away when you reboot. Static routes set by hand are not persistent by default. The kernel will "forget" everything unless you make this permanent. How do you make them permanent? This is, unfortunately, a bit more complicated. Different distributions set their network configurations differently - and have different tools for configuring the network configuration.

For example, if you're using Red Hat or Fedora, you'll find networking scripts under `/etc/sysconfig/network-scripts`, whereas Debian-based systems keep their information under `/etc/network/interfaces`. If you're using a desktop system, you might want to use Network Manager to make changes rather than using text files.

Users on openSUSE or SUSE Linux Enterprise Systems should use YaST2 to make changes. Bottom line, if you need to set up persistent routes, you'll probably want to consult your distribution's documentation.

Names to Numbers and Back

The Address Resolution Protocol (ARP) relates a host's IP address to the hardware address (MAC address) assigned to your network adapter. Historically, the ARP tables were read and managed by the `arp` command. You might not often need to touch `arp`, but it's handy to know you have the option of monitoring and managing the way your system handles address resolution.

Note that you only have ARP information about "neighbor" hosts on your local network. If you have a private *192.168.1.0/24* network, you can use `arp 192.168.1.71` and get something like Figure 3.

```
[jzb@f14 network-scripts]$ arp 192.168.1.71
Address                HWtype  HWaddress          Flags Mask          Iface
192.168.1.71           ether    d4:9a:20:57:6f:e4  C                   eth2
[jzb@f14 network-scripts]$
```

If you haven't pinged or interacted with the host previously, you won't have anything in the cache, so you can have two machines sitting next to one another on the network that have no ARP cache entries for their neighbors. If you ping the machine, you'll be able to get the ARP entry, which will include the MAC address under the *HWaddress* column.

Not surprisingly, the *ip* command provides a replacement. The object, in this case, is *neigh* for "neighbor."

Shorthand

One interesting and useful aspect of the `ip` command is that, when specifying an object, you do not need to type the entire object name. A number of the objects described in this article are abbreviations for the actual objects; for example, *address* is the object and *addr* is just an abbreviation, *neigh* is an abbreviation for *neighbor*, and so forth. The command `ip l` will show you the configured links just as `ip link` would. Note, however, that in a couple of cases multiple objects start with the same letter - for example, *address* and *addrlabel*. If you input just `ip a`, you are shown the addresses rather than the address labels. In general, the more common objects are recognized first. Also, you can use abbreviations for commands as well as objects.

The `arp` output for a specific host might look something like

```
192.168.2.67 ether 00:80:77:b8:1f:f6 C enp0s31f6
```

where the output from `ip neigh` would look like:

```
192.168.2.67 dev enp0s31f6  
lladdr 00:80:77:b8:1f:f6  
REACHABLE
```

Both commands output the IP address (*192.168.2.67*), the MAC address (*lladdr 00:80:77:b8:1f:f6*), and the network interface (*enp0s31f6*) that connects to this address.

Troubleshooting

Once you have finished configuring the network, you might need to check to ensure that packets can reach remote hosts. The `ping` command verifies that the networking system can successfully support communication with another computer on the network. You can specify either the hostname or the IP address:

```
ping 192.168.1.99
```

The output shows a report for each packet in an unending list that includes information on whether the attempt was successful or not, along with the response times. Although this continuous output can be useful for testing purposes, it is easily ended with `Ctrl` + `C`. To limit the number of packets, use the `-c` (count) option.

You might want to ping using a specific interface to try to troubleshoot networking problems. For example, if you have a server with two or more interfaces, you can specify the `enp0s31f6` interface to use with `ping -I enp0s31f6` (replace `enp0s31f6` with the name of the interface you'd like to use).

The `ping` command also allows you to set the interval between packets. The default is one second for each packet, or to send as fast as the system can with the `-f` (flood) option. Note that only root can use the flood option. To specify the interval, use `ping -i NN` where `NN` is the interval. This can be a fraction of a second, so if you want to send a ping every half second, use:

```
ping -i 0.5 192.168.1.99
```

Another option, short of using `flood`, is to preload the number of packets to be sent. This option will send a predetermined number of packets without waiting for a response. To send more than three, you'll need to use `sudo` or be root. The preload option is specified with `-l`, like so:

```
ping -l NN 192.168.1.99
```

Replace `NN` with the number of packets that you'd like to send.

Finally, you might want the Time to Live (TTL) option using the `-t` option. TTL is the maximum number of routers that a packet can travel before being thrown away.

Admins sometimes have the need to check the route a packet takes to its destination. Just because you can't reach a site doesn't mean the problem is on your network or the destination network - sometimes the problem is in between.

For example, say you can't reach *Woot.com* for some reason. It could be that *Woot.com* is down, or that you have a networking issue on your side. Or it might be that the problem lies between your network and *Woot.com*'s network, and one way to figure this out is by using utilities to trace the path that packets are taking.

The `traceroute` command and the newer `tracepath` utility provide this information. The `tracepath` utility is part of the `iputils` package that also includes `ping`. Although `traceroute` is the older utility, it has many more options than `tracepath`. In essence, the only thing you can pass to `tracepath` is a destination port number. On the other hand, `traceroute` allows you to specify time-to-live values, maximum hops, a specific interface to use, and many more options.

The basic syntax is simple enough: Use `traceroute host` and you'll see a listing of the hosts between your computer (or the system you are running `traceroute` on) and the final destination. Because you're using `traceroute` to check for overall latency and problems, if a host returns `***` but the packets are reaching their destination, this is OK.

The maximum TTL (number of hops) is usually set to 30. You might have more than 30 hops between yourself and the final host. To change this, use the `-m` option, like so:

```
traceroute -m 35 linuxpromagazine.com
```

This line would increase the number of hops to 35. Adjust if necessary.

Once again, you might need to use `traceroute` to debug specific interfaces on a machine. To do this, you can use `-i` (interface), `-s` (source address), or both options. A machine could have two or more IP addresses without actually having more than one interface, or each interface might have its own address. Therefore, if you want to specify an IP address on a system's second Ethernet interface, use:

```
traceroute -i enp0s31f6 -s 192.168.1.100
```

Naturally, you'll want to replace the IP address with the appropriate address. If the path of the packets is inefficient or unexpected, `route` or `ip route` will show you what routes are configured. Note that you only see the route configured from the local machine; it is very possible the problem might lie elsewhere.

Possibly a given router is explicitly configured **not** to provide any details. So, for example, `tracpath` might report "no reply." This situation does not mean you cannot connect to the target (which you can verify with `ping`); it simply means the intermediate router is not responding to the request from `tracpath` (or `traceroute`).

The `tracpath` documentation specifies that it is not a "privileged program" and can be executed by anyone. Although this is true, I have never had any trouble running `traceroute` as a normal user, except that it is usually not in a normal user search path.

Other troubleshooting utilities include the `netstat` command (which outputs information on connections, routing tables, and interface statistics) or the newer `ss` utilities. Although `ss` is part of the `iproute` package, its syntax is different from `ip`. See the `ss` man page for more information.

Troubleshooting DNS

The Domain Name System (DNS) translates the familiar alphanumeric domain names used in email addresses and web URLs (such as *linuxpromagazine.com* or *whitehouse.gov*) to and from the numeric IP addresses necessary for TCP/IP networking. As long as your system knows the location of a DNS server, this name resolution process happens invisibly; however, sometimes some troubleshooting is necessary. Also, sometimes for informational purposes, it is important to know the IP address associated with a domain name or the domain name associated with an IP address. A pair of classic Linux utilities that allow you to query the DNS system are `nslookup` and `host`, both of which are part of the `bind-utils` package. `nslookup` provides more functionality and more extensive output than `host`; however, `nslookup` is considered outdated and less able than some more modern equivalents.

A powerful and popular DNS tool in use today is `dig`, which is short for "domain information groper." Basically, `dig` performs a DNS lookup and then shows the results. The most basic use is

```
dig <hostname>
```

which should return quite a bit of output, including an answer section with the hostname and (by default) the IP address associated with the hostname.

But `dig` can tell you much more. For example, if you want to see what DNS servers the domain uses, run `dig NS hostname` - this command will return a list of DNS servers responsible for translating the domain name to an IP address.

In the output, you also see what DNS server(s) `dig` has used to perform its lookups. Here, I'm using Google's DNS:

```
;; Query time: 40 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Fri Feb 4 17:01:16 2011
;; MSG SIZE rcvd: 138
```

The SERVER is 8.8.8.8 - one of Google's public DNS servers. If you can't look up a hostname with your default servers, you can try using a different server by specifying it like this,

```
dig @8.8.4.4 www.linuxpromagazine.com
```

which tells `dig` to use the second Google DNS server.

When is this useful? For one thing, you can query a DNS server to see if it has information that hasn't propagated to your ISP's DNS servers. This information can be handy so that you can check to see whether you've changed DNS correctly before it propagates by querying the servers directly.

Finally, you can use `dig` to find any kind of DNS record. Want to see the MX (mail) hosts for a given domain? Use the `MX` directive:

```
dig MX linuxpromagazine.com
```

Putting It All Together

The `ping` utility is the simplest way of ensuring that you can reach a remote computer. If `ping` works, you can usually assume the network is configured correctly (or at least correctly enough for the packets to arrive at their destination). To be sure things are configured properly, check whether you can reach the remote machine with either the hostname or the IP address, the problem is most likely with DNS, so use the `dig` utility to try that out.

Interestingly enough, if you can reach it with the hostname, but not with the IP address, this often indicates a DNS problem as well (the entry for that host points to the wrong IP address).

If you cannot connect with either the hostname or IP address, the simplest approach is to start with the local machine and work your way outward.

The first question is whether the IP is configured correctly on the local system. To check the IP configuration, use `ip addr` (or `ip a` to be even lazier). Then `ping` the router to ensure that you are connected to it, and, if that works, `ping` another address beyond the router to ensure that the router is forwarding packets successfully. If you still haven't identified the problem, `traceroute` or `tracepath` should provide clues about where the packets are getting lost.

Tools for Supporting Wireless Networks

In recent years, wireless connectivity has become nearly automatic, with a few connection settings handled mostly through the GUI interface. However, it is certainly still possible to search for wireless networks and initiate connections at the command line.

The `iw` utility used to be the leading tool for text-based wireless configuration in Linux, but `iw` only works with the unsafe and obsolete Wired Equivalent Protocol (WEP). Most Linux systems today use (Wireless Protected Access) WPA supplicant [1] with the `wpa_cli` front end for terminal sessions. A new tool called the iNet Wireless Daemon (`iwd`) is a smaller and simpler tool that could one day replace WPA Supplicant as the standard option for command-line wireless configuration.

WPA Supplicant

Most distributions come with `wpa_supplicant` preinstalled. The standard configuration file is found in `/etc/wpa_supplicant/wpa_supplicant.conf`. If you don't have this file, Ubuntu users can find an example in `/usr/share/doc/wpa_supplicant/examples`. Use

```
sudo gunzip -k  
wpa_supplicant.conf.gz
```

to extract the compressed file, and then move it with the `mv` command to the `/etc/wpa_supplicant` directory.

The configuration file normally contains information about the control interface (`/var/run/wpa_supplicant`). Some distros also define a group like `netdev` or `wheel`, which means that members of those groups may execute the `wpa_cli` front end. Uncomment or add this line to give the `wpa_cli` front end permission to modify the file:

```
update_config=1
```

Next, start `wpa_supplicant` as a daemon in the background and define the wireless interface and the configuration file:

```
wpa_supplicant -B -i wlan0 -c
/etc/wpa_supplicant/wpa_supplicant.conf
```

Next, start `wpa_cli`. Figure 1 shows how to check the status of the program. You can scan for available networks with `scan` and show the results with `scan_results`. Use `add_network` to connect to a network from the results, or if you already know the SSID, you can enter the details. The networks are indexed numerically, so the first network is number 0:

```
> add_network 0
```

Finally, you can add and save your network credentials:

```
> set_network 0 ssid "CROCODILE"
> set_network 0 psk "crocodile123"
> enable_network 0
> save_config
> quit
```

To set the credentials, enable the network, and then save your configuration.

wpa_passphrase

Included in the WPA Supplicant package is `wpa_passphrase`, which modifies the configuration file and offers an alternative to `wpa_cli`. You can use this command to connect to a network with (E)SSID you already know:

```
wpa_passphrase CROCODILE
secret_passphrase
```

The tool prints a network section as it is used in the `/etc/wpa_supplicant/wpa_supplicant.conf` configuration file. The file may contain several of these blocks to define connections for more than one network and various security policies, including pre-shared keys (Listing 1).

Listing 1: /etc/wpa_supplicant/wpa_supplicant.conf

```
home network; hidden (E)SSID asteroid
network={
    ssid="CROCODILE"
    #psk="crocodile123"
    psk=855f5a29480465b0e54562dd9693435c108c314551d693cd0e36118f9c5d95d5
}
```

To add the output of the previous `wpa_passphrase` command to the configuration file, use the `>>` operator:

```
wpa_passphrase asteroid secret_passphrase >>
/etc/wpa_supplicant/wpa_supplicant.conf
```

Make sure you use `>>` rather than `>` to add the output to the configuration file, otherwise, you will overwrite the existing file. To help configuring your network connections, check out the `wpa_supplicant` man page or refer to the `wpa_supplicant.conf` example file if you have one.

iwd

WPA Supplicant has seen many improvements through the years, and, in general, it is much easier to connect Linux to a wireless network than it used to be. However, many experts believe that Linux wireless support is due for some reinvention. The world got a scare a few years ago, when WPA supplicant was shown to be susceptible to the KRACK attack on the WPA2 protocol [2]. Since then, KRACK vulnerabilities have been patched, and WPA3 has taken wireless security to a deeper level, but the complications in implementing a reliable solution underscored the inherent complexity and ungainliness of the WPA Supplicant codebase. That complexity, along with many dependencies, also means that WPA Supplicant is ill-suited for mobile devices and Internet of Things configurations. The need to simplify and provide a better solution for these technologies explains why efforts have been underway for several years to create a lean alternative to WPA Supplicant.

One alternative that has already arrived, although it still is not installed by default on most Linux systems, is the iNet wireless daemon (`iwd`) [3] [4]. Intel has been leading the development of `iwd`. In October 2019, the stable 1.0 version was released, and `iwd` continues to evolve. `NetworkManager` versions from 1.12.0 on can use `iwd` as their back end. `iwd` also works with alternatives such as `ConnMan` and `systemd-networkd`. And recently, a small GUI was released for users who want to do without `NetworkManager` or `ConnMan`. You can also access `iwd` from the command line using the `iwctl` command.

At this writing, `iwd` is still not installed by default on many Linux systems. If `iwd` is not on your Linux system, you'll need to take some preliminary steps (Listing 2). Line 1 in Listing 2 checks if `iwd` is already installed. You'll need to install `iwd` and then remove `NetworkManager` (line 4) and disable WPA (lines 5-7). Finally, enable `iwd` (lines 8 and 9), and check to see if everything is working (line 10).

Listing 2: First Steps (look at Figure 2 on page 81 in Linux Shell Handbook - 2021 Edition)

```
01 $ systemctl status iwctl.service
02 Unit iwctl.service could not be found.
03 $ sudo apt install iwctl
04 $ sudo apt purge network-manager
05 $ sudo systemctl stop wpa_supplicant.service
06 $ sudo systemctl disable wpa_supplicant.service
07 $ sudo systemctl mask wpa_supplicant
08 $ sudo systemctl enable iwctl.service
09 $ sudo systemctl start iwctl.service
10 $ systemctl status iwctl.service
```

It is a bad idea to remove the *wpa_supplicant* package after the preliminary work is complete, instead of just disabling it. On Ubuntu, removing *wpa_supplicant* would also remove the *ubuntu-desktop* metapackage due to many dependencies. On Debian, `NetworkManager` would be removed as well - which might be a benefit in some cases.

WLAN Setup

Once you have completed the necessary steps, and assuming the status query is positive, you can setup WiFi access. If you get a message about `rfkill` blocking (Figure 2), call the command:

```
sudo rfkill list wifi
```

If *Soft blocked* shows up as *yes*, pressing `Fn` + `F5` might help to switch off flight mode. If this does not help, use:

```
sudo rfkill unblock wifi
```

Check if this worked with `rfkill` or a new status request for `iwctl.service`.

Now launch an interactive shell as a normal user with the `iwctl` command. Typing *help* lists all the available options. To exit the shell, press `Ctrl` + `D`. `iwctl` can also be used without an interactive shell; you just have to prefix each command with `iwctl`. Use `device list` to discover the name the system is using for the interface (Figure 3). If the interface goes by the name of *wlan0*. The command

```
device wlan0 show
```

delivers more details about the network interface card. Now scan by typing `station wlan0 get-networks` to display the available networks.

Enter the following command:

```
station device_name connect network_name
```

to enable the connection. The requested password is stored in `/var/lib/iwctl` when input with the `.psk` suffix.

If needed, check the functionality again by typing:

```
status device_name get-networks
```

A check mark, hardly visible against the dark color scheme of the Ubuntu terminal, indicates that the connection was successfully opened. Then use `ping` to check the status of the Internet connection or browse to a website. After rebooting the computer, `iwd` automatically re-establishes the wireless connection.

If the connection fails, or if problems occur while roaming through changing networks, create a `/etc/iwd/main.conf` file with the content from Listing 3.

Listing 3: main.conf

```
[General]
EnableNetworkConfiguration=true
[Network]
NameResolvingService=systemd
```

The configuration causes `iwd` to hand over name resolution to `systemd-resolved`. `resolvconf` is also available as an alternative.

Using Samba (setting up a Samba share)

Samba shares allow you to transfer files and directories between Linux systems and non-Linux systems as if they were mounted locally to your Linux system. It took me four days of work (summer of 2021) to figure this out. This setup procedure may not be flawless, but it worked pretty well for me at the time. One thing that I've learning about troubleshooting computers over the years is "Just because it worked on somebody else's system, doesn't necessarily mean that it will work on my system." Keep that in mind. If this procedure works for you without a hitch, that's great. Otherwise, get to googling and figure it out for yourself. Linux is all about the "trial and error" heuristic. Hence the quote from programmer Jamie Zawinski, co-founder of the Mozilla Project: "Linux is only free if your time has no value." Forget that noise and work the problem.

On a Linux machine (source: <https://ubuntu.com/server/docs/samba-file-server>)

1. `sudo apt install samba`
2. Edit the following key/value pairs in the [global] section of `/etc/samba/smb.conf`.
 1. `sudo nano /etc/samba/smb.conf`
 2.

```
workgroup = WORKGROUP
security = user
wins support = yes
```
3. Under Share Definitions: (`#` initiates comment in bash)

```
[share]
comment = Ubuntu File Server Share
path = /srv/samba/share
browsable = yes
writable = yes
guest ok = yes
create mask = 0777 # 0777 (octal) == binary 0b 111 111 111 ==
permissions rwxrwxrwx (== decimal 511)
force user = <insert your user here>
```

Save and exit `smb.conf` file.

3. Create a directory for the mountpoint (could be created in `/mnt`): `sudo mkdir -p /srv/samba/share`

```
4. Change the ownership permissions: `sudo chown nobody:nogroup /srv/samba/share/`
5. Restart the samba services to enable the new configuration: `sudo systemctl restart smbd.service nmbd.service`
6. `sudo mount -t cifs //192.168.1.131/shared_directory /srv/samba/share/`
```

On a Windows machine

1. Run `appwiz.cpl`, and enable the SMB protocol.
2. Type the IP address of Samba server in the bar at the top.
3. `smb://192.168.1.131`

On a macOS machine

1. Assuming the services are functioning as expected, the ID for your Samba server should appear under the word "Shared" in the sidebar to the left whenever you open Finder.
2. You might have to type `smb://192.168.1.131` in the bar at the top.

Using WINE

WINE stands for "**WINE IS NOT AN EMULATOR**". It allows you to run Windows applications on a Linux platform. As the name suggests, it is not an emulator. I think of WINE as a sort of translator, designed to port (adapt) Windows application binary interfaces (ABIs) to Linux ABIs. From what I understand, ABIs are pretty much only relevant as far as software developers are concerned. End users, by contrast, utilize application programming interfaces (APIs) in order to retrieve data from a server.

Preliminary Steps for installing WINE versions 5.0 & 6.0 (source: <https://wiki.winehq.org/Ubuntu>)

1. `sudo dpkg --add-architecture i386`
2. `sudo apt update`
3. `sudo apt install wine64 wine32`
4. `wine --version`

Procedure for installing WINE version 6.0 (sources: <https://wiki.winehq.org/Ubuntu> and

<https://linuxhint.com/install-winehq-on-ubuntu-20-04/>

1. `wget -nc https://dl.winehq.org/wine-builds/winehq.key`
2. `sudo apt-key add winehq.key`
3. `hostnamectl | grep Operating`
4. (May not even be necessary, but might be): `sudo apt install software-properties-common`
5. `sudo apt-add-repository 'deb https://dl.winehq.org/wine-builds/ubuntu/ impish main'`
6. `sudo apt update`
7. `sudo apt install --install-recommends winehq-stable`
8. `wine --version`
9. `winecfg`
10. Install the wine-mono package and Gecko.
11. `wget https://github.com/notepad-plus-plus/notepad-plus-plus/releases/download/v7.8.8/npp.7.8.8.Installer.exe`
12. `wine npp.7.8.8.Installer.exe`
13. After selecting a language and clicking **“OK”**, the **“Notepad ++”** installer will appear. To continue, click **“Next”**.
14. Choose the location where you want to save this file; click **“Next”** to continue.
15. Click **“Install”**.
16. Once the installation is completed, Hit **“Finish”**.
17. To launch the **“Notepad++”**, type **“notepad++”** in the applications search bar, or type `cd ~/.wine/drive_c/'Program Files'/Notepad++;wine notepad++.exe` to change to the installation directory at the command line and open Notepad++ from the command line.
18. Alternatively, you can open **“notepad++”** by using the command line. `wine notepad++`
19. A preloaded file will open.
20. Notepad has successfully been installed on the Linux system.
21. Using the **“purge”** command, you can remove the **“wineHQ”** application and all of its packages from your system:
`sudo apt purge winehq-stable`
22. **“WineHQ”** has been removed from your system.

Cory's Favorite Terminal Color Scheme

```
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=
40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:
ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:
*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01
;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;3
1:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=0
1;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar
=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.
rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:
*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm
=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.
png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;
35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.webp=01;35:*.ogm=01;35:*.m
p4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:
*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;
35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01
;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid
=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.
wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
```

Bright neon colors are easier for me to see.

I used Hex codes from this website: (https://www.fashiontrendsetter.com/content/color_trends/color-decoder/Color-Code-Neon.html)

Desktop Environments & Window Managers

Typically, a desktop environment (DE) runs in the X Window System (aka X11 or just X). X was developed by X.Org, a non-profit foundation chartered to research, develop, support, organize, administrate, standardize, promote, and defend a free and open accelerated graphics stack. This includes, but is not limited to, the following projects: DRM, Mesa 3D, Wayland and the X Window System (in the implementation of the X.Org Server).

A window manager provides decorative and functional borders around X windows. When you resize or drag a window, it's the window manager that you're using. Most window managers also control the root window--that is, the screen as a whole. If you right-click the desktop's background, chances are you'll see a menu pop up; that's a window manager tool. Common window managers include dwm, fwm, twm, IceWM, Blackbox, and Metacity.

Some minimalist Linux users run a window manager in their X sessions but little else, aside from whatever programs they actively use. Most users, though, run a desktop environment atop the window manager. This is a set of software tools that facilitates launching programs, adjusting user interface settings, and so on. Desktop environments also typically include a file manager, which provides drag-and-drop file manipulation. The most popular Linux desktop environments are the GNU Network Object Model Environment (GNOME), the K Desktop Environment (KDE), and Xfce.

Programs that I've Installed on Linux machines (successfully or not)

- Vim
- snapd
- Kdenlive
- graphicsmagick-imagemagick-compat

- GNU Privacy Guard (gpg)
- Virtualbox
- Samba
- pavucontrol pulseaudio-module-gconf pulseaudio-module-zeroconf
- cifs-utils
- network-manager
- network-scripts
- ifupdown
- wpa_supplicant
- iwid
- build-essential wget unzip openssl libssl-dev libgd2-xpm-dev xinetd apache2 php\apache2-utils apache2-mod-php7.0 php-gd
- policycoreutils
- remarkable
- perl-doc
- printf
- wmctrl
- htop
- net-tools
- SpeechRecognition pydub (using pip3)
- python3-pip
- simple-scan
- youtube-dl
- traceroute
- git
- haskell-stack
- curl
- cabal
- graphviz
- erd
- nix-build
- software-properties-common
- vlc
- kodi
- pdfshuffler
- pdftk
- ffmpeg
- wine
- wine-stable winehq-stable
- wine64 wine32
- --install-recommends winehq-stable
- smbclient
- vokoscreenNG
- pandoc
- screenfetch
- inxi
- lsscsi

Builtin Commands That Are Symbols

The Bourne Again Shell uses the symbols `(,), [,],` and `$` in a variety of ways.

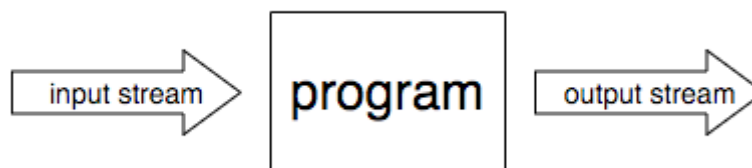
Symbol	Command
<code>()</code>	Subshell (page 344)
<code>\$()</code>	Command substitution (page 410)
<code>(())</code>	Arithmetic evaluation; a synonym for <code>let</code> (use when the enclosed value contains an equal sign; page 1056)
<code>\$(())</code>	Arithmetic expansion (not for use with an enclosed equal sign; page 408)
<code>[]</code>	The <code>test</code> command (pages 983 and 986)
<code>[[]]</code>	Conditional expression; similar to <code>[]</code> but adds string comparisons (page 1057)

I/O Streams & Redirection

Streams are just that: streams. **In the same way that a river has a stream of water, programs have streams of data.** Moreover, **just like you can use steel pipes to carry water from one place to another, you can use UNIX pipes to carry data from one program to another.** This was the very analogy that inspired the design of streams:

We should have some ways of connecting programs like a garden hose — screw in another segment when it becomes necessary to massage data in another way. This is the way of I/O also. — Douglas McIlroy

Streams can be used to pass data into programs and to get data out of them.

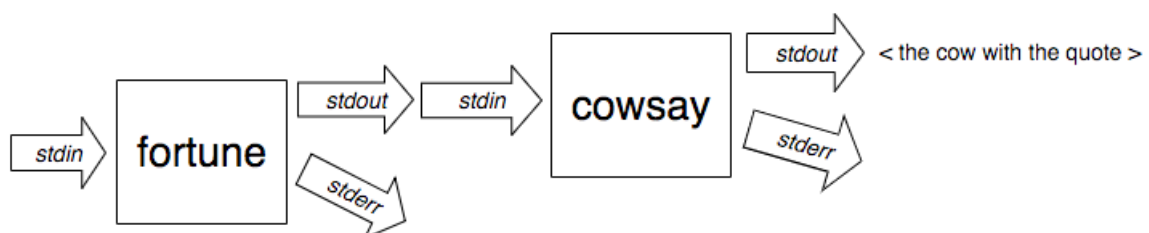


In UNIX, programs get some streams attached to them by default, both for input and output. We call these *standard streams*.

There are three different standard streams:

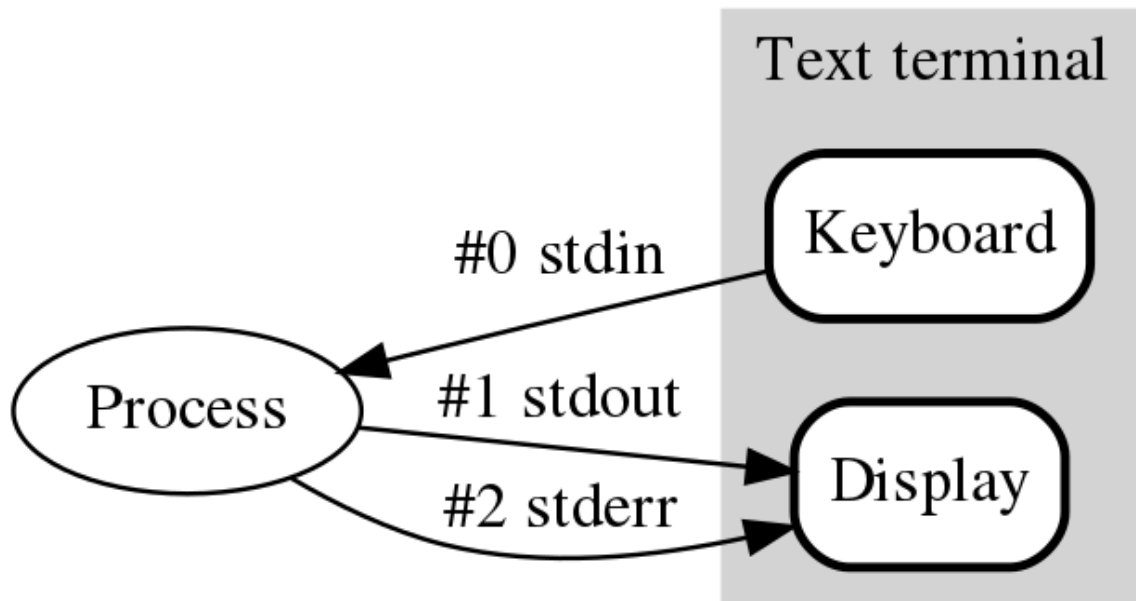
- `stdin` or *standard input* is the stream which feeds your program with data
- `stdout` or *standard output* is the stream your program writes its main output to
- `stderr` or *standard error* is the stream your program writes its error messages to

We use pipes for connecting the output stream of a program to the input stream of another.



You can see the output of `cowsay` in your screen because, by default, your terminal gets the `stdin`, `stdout` and `stderr` standard streams attached to it.

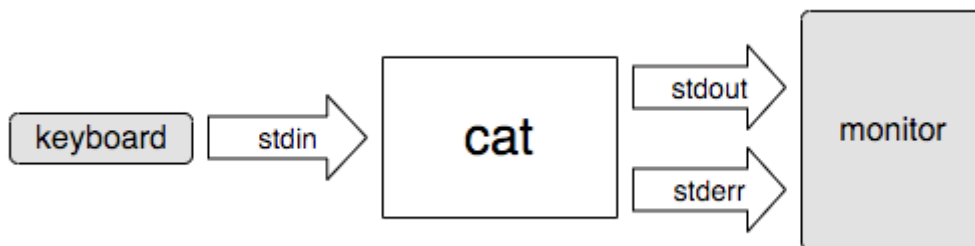
Data goes in through `stdin` and `stderr` and goes out in the other end: your monitor. Similarly, your keyboard input goes through `stdin` to a program.



Source: [Wikipedia](#)

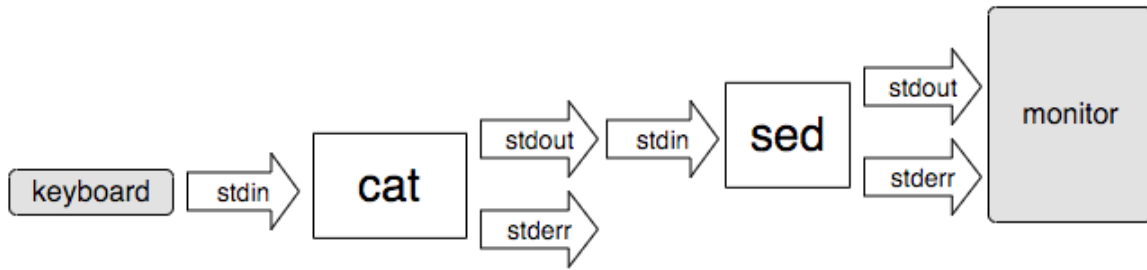
The `cat` program, for example, uses the `stdin` to receive input from your keyboard and the `stdout` to send it out:

```
$ cat
Everything I write before pressing Enter
Everything I write before pressing Enter
Gets logged right after
Gets logged right after
```



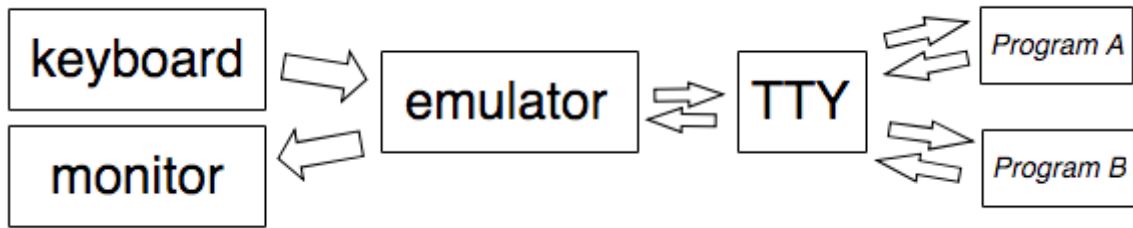
We can make it more elaborate by using `sed` to replace all occurrences of `I` by `we` every time we press `Enter`:

```
$ cat | sed -E "s/I/we/"
I think streams are quite cool.
We think streams are quite cool.
```



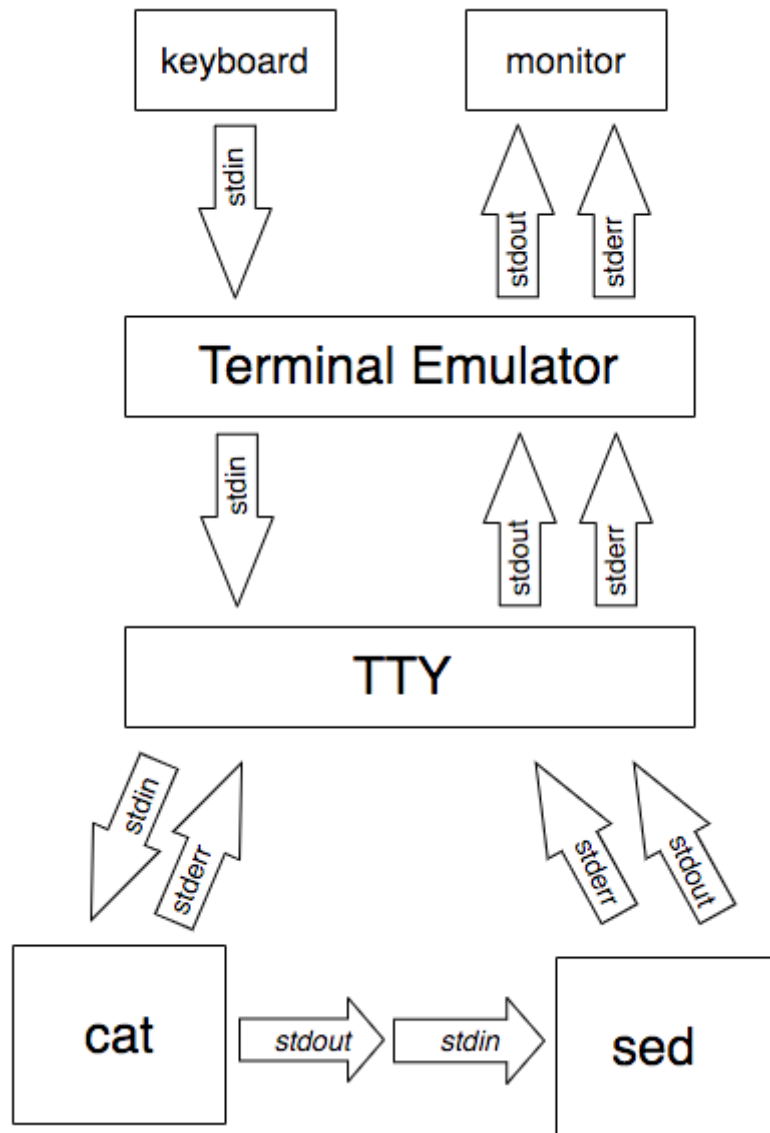
Also, in case you didn't know, `sed` stands for `stream editor`.

Terminal emulators are software simulations of "real" terminals. These emulators provide you with an interface to interact with the Linux TTY driver. **The TTY driver is responsible for handling the data to and from programs.**



Each TTY has its own `stdin`, `stdout`, and `stderr` streams connected to it. These are the streams provided to programs for them to read from (`stdin`) and write to (`stdout` and `stderr`).

Here is a more accurate version of what happened when you ran `cat | sed -E "s/I/We/"` in the last example:



[Like everything in UNIX](#), the `tty` is a file. Each instance of a terminal emulator has a different `tty` file associated with it. Because each emulator reads from and writes to a different file, you don't see the output of the programs you run in all the windows you have open.

To find out which `tty` is associated with a terminal window you can use the `tty` command.

Pipes and redirects allow output to be either piped to another command or redirected to a file.

- `|`
 - Pipe output from one command to another
- `ls | tee peanuts.txt`
 - lists contents and sends contents to a file. (like 'T' shaped splitter in plumbing)
- `2>&1` or `&>`
 - Redirect error to destination file.

Syntax	visible in terminal		visible in file		existing
	StdOut	StdErr	StdOut	StdErr	file
>	no	yes	yes	no	overwrite
>>	no	yes	yes	no	append
2>	yes	no	no	yes	overwrite
2>>	yes	no	no	yes	append
&>	no	no	yes	yes	overwrite
&>>	no	no	yes	yes	append
tee	yes	yes	yes	no	overwrite
tee -a	yes	yes	yes	no	append
n.e. (*)	yes	yes	no	yes	overwrite
n.e. (*)	yes	yes	no	yes	append
&& tee	yes	yes	yes	yes	overwrite
&& tee -a	yes	yes	yes	yes	append

For the supposedly "non-existent" entries, you could use:

```
{ <command> 2>&1 >&3 | tee <filename> >&2; } 3>&1
```

This supposedly works in all POSIX shells.

Filters

One kind of program frequently used in pipelines is called a *filter*. Filters take standard input and perform an operation upon it and send the results to standard output. In this way, they can be combined to process information in powerful ways. Here are some of the common programs that can act as filters:

Program	What it does
<code>sort</code>	Sorts standard input then outputs the sorted result on standard output.
<code>uniq</code>	Given a sorted stream of data from standard input, it removes duplicate lines of data (i.e., it makes sure that every line is unique).
<code>grep</code>	Examines each line of data it receives from standard input and outputs every line that contains a specified pattern of characters.
<code>fmt</code>	Reads text from standard input, then outputs formatted text on standard output.
<code>pr</code>	Takes text input from standard input and splits the data into pages with page breaks, headers and footers in preparation for printing.
<code>head</code>	Outputs the first few lines of its input. Useful for getting the header of a file.
<code>tail</code>	Outputs the last few lines of its input. Useful for things like getting the most recent entries from a log file.
<code>tr</code>	Translates characters. Can be used to perform tasks such as upper/lowercase conversions or changing line termination characters from one type to another (for example, converting DOS text files into Unix style text files).
<code>sed</code>	Stream editor. Can perform more sophisticated text translations than <code>tr</code> .
<code>awk</code>	An entire programming language designed for constructing filters. Extremely powerful.

System Administration

UNIX Job Control

The following are part of bash, not separate utils, so there's no man page for them. They are documented in **man bash**.

- `cmd &`
 - Start a program in background. For example, `ruby process_log.rb &`
- `Ctrl + C`
 - Stop the current program associated with your terminal. (sending SIGINT to it)
- `Ctrl + Z`
 - Suspend the current program associated with your terminal. (sending SIGTSTP to it)
- `jobs`
 - List background processes
- `bg %number`
 - Run the suspended command in background.
- `fg %number`
 - Resume a background process in foreground.
- `disown %number`
 - Separate a job ID `number` from jobs table.

A UNIX process signal is an integer sent to a program to tell it something. To see a list of all signals, type `man kill`.

1. Start `xclock` with a second hand by `xclock -update 1`, it'll hog the terminal.
2. Now, in terminal, press `Ctrl+z` to pause the program and returns you the prompt. (when the program is paused, it can't be used. You'll see that the second-hand stopped moving.)
3. Type `jobs` to list all jobs and their ID in job table. Here's a sample output:

```
[1]+  Stopped                  xclock -update 1
```
4. Type `bg %1` to start the process with job ID 1 in background.
5. Now, it is as if you started xclock by `xclock &`.

The "xclock" example above can be any shell command.

Here's some other commands that are very useful. Note, many GUI apps these days will detach itself from terminal, even if you didn't start it with `&`.

- `setsid <command>`
 - Run a program in a new session.
- `nohup <command>`
 - Run a command immune to `SIGHUP` signal, and redirect stdout to a normal file.

Closing Terminal Kills My Program?

Tips:

- Don't close your terminal by clicking the window close. Close it by `Ctrl+d`. Because, when you click on close, it may kill all programs you launched from it.
- if you do `emacs &`, then close your GUI terminal (by clicking on the close button), it'll also take down your emacs.
- You can use `setsid emacs` to launch program. It runs the program in its own session.

To run a server, you can do: `nohup myserver > out.txt 2> err.txt < /dev/null &`.

This will set stdout to out.txt, and stderr to err.txt, and stdin from `/dev/null`, and not receive signal `SIGHUP` signal, and remove this process from jobs table.

Processes

A **process** like we said before is a running program on the system, more precisely it's the system allocating memory, CPU, I/O to make the program run. A process is an instance of a running program.

The kernel is in charge of processes, when we run a program the kernel loads up the code of the program in memory, determines and allocates resources and then keeps tabs on each process, it knows:

- The status of the process
- The resources the process is using and receives
- The process owner
- Signal handling (more on that later)
- And basically everything else

All processes are trying to get a taste of that sweet resource pie, it's the kernel's job to make sure that processes get the right amount of resources depending on process demands. When a process ends, the resources it used are now freed up for other processes.

When a new process is created, an existing process basically clones itself using something called the fork system call (system calls will be discussed very far into the future). The fork system call creates a mostly identical child process, this child process takes on a new process ID (PID) and the original process becomes its parent process and has something called a parent process ID PPID. Afterwards, the child process can either continue to use the same program its parent was using before or more often use the execve system call to launch up a new program. This system call destroys the memory management that the kernel put into place for that process and sets up new ones for the new program.

We can see this in action:

```
$ ps l
```

The l option gives us a "long format" or even more detailed view of our running processes. You'll see a column labelled PPID, this is the parent ID. Now look at your terminal, you'll see a process running that is your shell, so on my system I have a process running bash. Now remember when you ran the ps l command, you were running it from the process that was running bash. Now you'll see that the PID of the bash shell is the PPID of the ps l command.

So if every process has to have a parent and they are just forks of each other, there must be a mother of all processes right? You are correct, when the system boots up, the kernel creates a process called init, it has a PID of 1.

The init process can't be terminated unless the system shuts down. It runs with root privileges and runs many processes that keep the system running.

We will take a closer look at init in the system bootup course, for now just know it is the process that spawns all other processes.

A process can exit using the `_exit` system call, this will free up the resources that process was using for reallocation.

So when a process is ready to terminate, it lets the kernel know why it's terminating with something called a termination status.

Most commonly a status of 0 means that the process succeeded. However, that's not enough to completely terminate a process.

The parent process has to acknowledge the termination of the child process by using the wait system call and what this does

is it checks the termination status of the child process. I know it's gruesome to think about, but the wait call is a necessity,

after all what parent wouldn't want to know how their child died?

There is another way to terminate a process and that involves using signals, which we will discuss soon.

Orphan Processes

When a parent process dies before a child process, the kernel knows that it's not going to get a wait call,

so instead it makes these processes "orphans" and puts them under the care of init (remember mother of all processes).

Init will eventually perform the wait system call for these orphans so they can die.

Zombie Processes

What happens when a child terminates and the parent process hasn't called wait yet? We still want to be able to see how a child process terminated, so even though the child process finished, the kernel turns the child process into a zombie process. The resources the child process used are still freed up for other processes, however there is still an entry in the process table for this zombie. Zombie processes also cannot be killed, since they are technically "dead", so you can't use signals to kill them. Eventually if the parent process calls the wait system call, the zombie will disappear, this is known as "reaping". If the parent doesn't perform a wait call, init will adopt the zombie and automatically perform wait and remove the zombie. It can be a bad thing to have too many zombie processes, since they take up space on the process table, if it fills up it will prevent other processes from running.

A signal is a notification to a process that something has happened.

Why we have signals

They are software interrupts and they have lots of uses:

- A user can type one of the special terminal characters (Ctrl-C) or (Ctrl-Z) to kill, interrupt or suspend processes
- Hardware issues can occur and the kernel wants to notify the process
- Software issues can occur and the kernel wants to notify the process
- They are basically ways processes can communicate

Signal process

When a signal is generated by some event, it's then delivered to a process, it's considered in a pending state until it's delivered.

When the process is ran, the signal will be delivered. However, processes have signal masks and they can set signal delivery to be blocked if specified.

When a signal is delivered, a process can do a multitude of things:

- Ignore the signal
- "Catch" the signal and perform a specific handler routine
- Process can be terminated, as opposed to the normal exit system call
- Block the signal, depending on the signal mask

Common signals

Each signal is defined by integers with symbolic names that are in the form of SIGxxx. Some of the most common signals are:

- `SIGHUP` or `HUP` or 1: Hangup
- `SIGINT` or `INT` or 2: Interrupt
- `SIGKILL` or `KILL` or 9: Kill
- `SIGSEGV` or `SEGV` or 11: Segmentation fault
- `SIGTERM` or `TERM` or 15: Software termination
- `SIGSTOP` or `STOP`: Stop

Numbers can vary with signals so they are usually referred to by their names.

Some signals are unblockable, one example is the `SIGKILL` signal. The `KILL` signal destroys the process.

You can send signals that terminate processes, such a command is aptly named the kill command.

```
$ kill 12445
```

The 12445 is the PID of the process you want to kill. By default it sends a TERM signal. The SIGTERM signal is sent to a process to request its termination by allowing it to cleanly release its resources and saving its state.

You can also specify a signal with the kill command:

```
$ kill -9 12445
```

This will run the SIGKILL signal and kill the process.

Differences between SIGHUP, SIGINT, SIGTERM, SIGKILL, SIGSTOP?

These signals all sound reasonably similar, but they do have their differences.

- `SIGHUP` - Hangup, sent to a process when the controlling terminal is closed. For example, if you closed a terminal window that had a process running in it, you would get a SIGHUP signal. So basically you've been hung up on
- `SIGINT` - Is an interrupt signal, so you can use Ctrl-C and the system will try to gracefully kill the process
- `SIGTERM` - Kill the process, but allow it to do some cleanup first
- `SIGKILL` - Kill the process, kill it with fire, doesn't do any cleanup
- `SIGSTOP` - Stop/suspend a process

When you run multiple things on your computer, like perhaps Chrome, Microsoft Word or Photoshop at the same time, it may seem like these processes are running at the same time, but that isn't quite true.

Processes use the CPU for a small amount of time called a time slice. Then they pause for milliseconds and another process gets a little time slice. By default, process scheduling happens in this round-robin fashion. Every process gets enough time slices until it's finished processing. The kernel handles all of these switching of processes and it does a pretty good job at it most of the time.

Processes aren't able to decide when and how long they get CPU time, if all processes behaved normally they would each (roughly) get an equal amount of CPU time. However, there is a way to influence the kernel's process scheduling algorithm with a nice value. Niceness is a pretty weird name, but what it means is that processes have a number to determine their priority for the CPU. A high number means the process is nice and has a lower priority for the CPU and a low or negative number means the process is not very nice and it wants to get as much of the CPU as possible.

One Big Family

Processes are never isolated and are always in good company. In fact, they are in a hierarchical structure, with process number 1, *init* at the top. On most distributions, *init* has been replaced by *systemd*, and `/sbin/init` is a symlink. This is the first process that Linux launches after booting. All other processes share this common "ancestor" - *systemd* starts the operating system's basic programs. Entering `ps f` presents you with a tree view of processes in the form of an ASCII image. As an alternative, you can run the `pstree` program, which also gives you a useful overview of the relationships between "parent" and "child" processes. This tree structure shows you at a glance who is descended from whom.

The `pstree` tool gives you more detailed output if you set the `-a` flag. This tells `pstree` to show you the parameters with which the programs are running. If you use a terminal that supports different fonts and bold type, such as Gnome Terminal or KDE's Konsole, you might also want to try the `-h` parameter. This tells `pstree` to highlight its own process and its ancestors. If you want to use this practical feature for other processes, use `-H` with the process ID, and `pstree` will highlight the specified process and its family tree. Setting the `-p` option tells `pstree` to output the process ID (PID), and `-u` shows the user. All of these parameters can be combined - for example, `pstree -apuh`.

Top Tool!

If you are looking for CPU hogs, `ps` is not your best option. Because it simply gives you a snapshot of the current status, you will not find out too much about the current system load. However, Linux has the *top* tool to help you with this task. *Top* is a process monitor that updates the display to give you the current status. You can launch the monitor by typing `top` at the command line.

This program gives you extensive information about your system and the processes running on it. The top line shows the time, the computer uptime, the number of processes, and the status details, along with the CPU, memory, and swap load. To find out more about used and unused memory and swap space, you can also use `free` or `uptime`.

The *top* status line contains information on the individual processes. The columns of the status line present various categories:

- process ID (PID)
- username (USER)
- Priority (PR)
- nice level (NI)
- memory usage as a percentage (%MEM)
- parent process ID (PPID)
- user ID (UID)
- CPU time consumed as a percentage (%CPU)
- command name (COMMAND)

You also can tell *top* what you want to see; just press `F` and the relevant letters to specify the status line content. Several commands allow you to control *top* interactively; for example, you can press `H` to display the online help. Entering `U` followed by a username gives you the processes for that user. `Shift` + `R` reverts the output, showing the most frugal processes instead of the CPU hogs, and entering `Q` quits the tool and takes you back to the shell. `Shift` + `Z` lets you add color. The `W` key toggles through several predefined color schemes, but you can also press the appropriate letters and numbers to define your own color scheme.

Mister Nice Guy

Processes have a specific priority, which becomes useful if you have a program running in the background and do not want to risk losing control over the system load. To start a program with a specific priority, use the `nice` command. Non-privileged users may only assign lower priorities to their own tasks - assigning higher priorities is the administrator's domain.

Processes have a `nice` value of 0 by default; if you run the command without parameters, the program will confirm:

```
$ nice
0
```

With the `nice` command, you can also assign a specific priority, where `**`-20 is the highest and 19 is the lowest priority. To set the level for the process monitor `top`, for example, you would type:

```
nice -n 19 top
```

If you skip the `-n` option and the `nice` level, `nice` sets the level to 10. As mentioned before, regular users are only allowed to use positive increments:

```
$ nice -n -19 top
nice: cannot set niceness:
Permission denied
```

To discover a program's nice level, use the `top` (under column NI in the status bar) or `ps` command. In the `ps` output in Listing 3, the `top` call has been "niced," as indicated by the capital N in the `STAT` column.

To change the priorities of programs that are already running, use the `renice` command. Regular users manipulate only their own tasks; only the root user can `renice` every program. To change a priority, find out the program's PID, then use `renice` plus the `-n` parameter and the value:

```
$ renice -n 10 2342
2342 (process ID) old priority 19, new priority 10
```

More Information About Memory

Two more command-line tools provide information on your system's memory. The `free` command is typically used without any parameters and shows the total amount of free and used physical and swap memory in the system, as well as the buffers used by the kernel. The default setting is kilobytes; to display megabytes or gigabytes, use the `-m` or `-g` switch. It's also possible to add a fourth line of data containing the totals for physical memory and swap space. To do so, use the `-t` option. To find out more about your system's load averages, you can use the `uptime` tool. As the name suggests, this command is mainly used to show how long the system has been running. The output additionally displays the current time, the uptime, how many users are currently logged on, and the system's load averages for the past 1, 5, and 15 minutes.

```
$ uptime
17:49:04 up 30 min,  2 users,  load average: 0.00, 0.01, 0.08
```

Uptime Load Averages

To understand the load average numbers, you need to know how many CPUs (or how many CPU cores) your system has.

For a system with one CPU, a load average greater than 1.00 means the CPU is overloaded by whatever percentage for that period of time (1 minute, 5 minutes, or 15 minutes, respectively). But if the load average is less than 1.00 for that same system, it means that the CPU was idle by whatever percentage for that time period.

Cron and At keep your tasks on task

The Linux environment includes a number of utilities that allow you to schedule tasks. Two classic Bash scheduling tools are the At program, which lets you schedule tasks right now, and Cron, which handles recurring jobs. A daemon runs in the background to ensure the tasks are performed according to the schedule and checks for new jobs once a minute. The daemon for At is named **atd**, and the cron daemon is called **cron(d)**.

Systemd has introduced a new method for scheduling tasks in Linux: a Systemd timer. Like many features of the Systemd landscape, timers are relatively new, and many users still prefer to schedule tasks the old way. This article focuses on the classic Cron and At tools.

At Your Service

To perform a job, call `at` with the time at the command line, type commands in the shell, and quit by pressing `Ctrl + D`:

```
$ at 07:00
warning: commands will be executing using /bin/sh
at> ogg123 -ZZ /home/huhn/music/*
at> <EOT>
job 1 at Tue Nov 10 07:00:00 2009
```

This tells the `ogg123` command-line player to wake you on the dot at 7am by playing a random selection of songs in shuffle mode from the `/home/huhn/music/` directory - of course, this assumes your computer is switched on.

Table 1 gives an overview of the most common notations for time. Note that `at` is persistent: that is, it will keep running after you reboot your machine.

After completing a task, At sends email with the job status to the job owner as to whether the job completed successfully or not. Therefore, you need a working mail server configuration (at least for local deliveries). For commands that do not create output by default (e.g., `rm`, `mv`, or `cp`), you can enforce an email message. To do so, set the `-m` flag, as in `at -m 13:31`.

Displaying and Deleting Jobs

Scheduled At commands are stored in the queue, and you can display the queue onscreen by calling `at -l` or `atq`:

```
$ atq
2 Tue Nov 9 16:22:00 2010 a huhn
3 Tue Nov 10 17:08:00 2009 a huhn
4 Tue Nov 10 17:10:00 2009 a huhn
```

Unfortunately, `at` is not very talkative; it just tells you the job number, date and time, queue name (*a*), and username. The list does not tell you what jobs are scheduled. Additionally, you only get to see your own jobs as a normal user; only the system administrator gets to see a full list of scheduled jobs.

If you want more details on what the future holds, become root and change to the At job directory below `/var/spool`, for example `/var/spool/atjobs` (for openSUSE) or `/var/spool/cron/atjobs/` (for Debian and Ubuntu). The text files tell you exactly what commands will be run. Here you can also learn the user and group IDs (see the "Users and Groups" article) and the username of the one who started the `at` command.

To delete an At job, enter `at -d` or `atrm`, specifying the job number:

```
$ atrm 2 3
$ atq
4 Tue Nov 10 17:10:00 2009 a huhn
```

Access Privileges

Two files, `/etc/at.allow` and `/etc/at.deny`, control who is permitted to work with At. Most distributions tend just to have an **at.deny** file with a few "pseudo-user" entries for **lp** (the printer daemon) or **mail** (for the mail daemon). If you create an **at.allow** file as root, you need entries for all users who are permitted to run At jobs - **at.deny** is not parsed in this case.

Users who are not listed in **at.allow** therefore receive the message **You do not have permission to use at.**

A cron (command run on (UNIX) scheduler) for All Seasons

If you are looking for a way to handle regularly recurring tasks, repeatedly running At is not recommended. Instead you should investigate the other option that Linux gives you. Cron also runs in the background and runs jobs at regular intervals. Again, the `cron` program just needs the machine to be up because it "remembers" scheduled jobs when you reboot your machine. In fact, the two programs have even more in common: just like `at`, `cron` mails the owner account to confirm that a job has completed successfully. (Remember that this requires a working mail server and at least local delivery.)

Individual tasks are referred to as cronjobs, and they are managed in the crontab. This is a table with six columns that defines when a specific job is to be performed. Each command in the crontab occupies a single line. The first five fields describe the time, whereas the sixth field contains the program to be run, including any parameters.

As a normal user, you can create a crontab at the command line by running the crontab program with `crontab -e`, where the parameter `-e` indicates that you will be editing the table.

As the system administrator, you can additionally modify the crontabs of any user by specifying the `-u` parameter and supplying the account name:

```
crontab -u huhn -e
```

Usually, this calls the Vi editor - if you prefer a different text editor, just set your `$EDITOR` environment variable to reflect this, as in:

```
export EDITOR=/usr/bin/gedit
```

To make this change permanent, add this line to your Bash configuration file, and reparse the configuration file by entering `source ~/.bashrc`.

Well Structured

Crontab lines are not allowed to contain line breaks. Six fields contain the following information in this order:

- (1) minutes (0 to 59 and the * wildcard)
- (2) hour (0 to 23 or *)
- (3) day (1 to 31 or *)
- (4) month (1 to 12, Jan to Dec, jan to dec, or *)
- (5) weekday (0 to 7, where both 0 and 7 mean Sunday, Sun to Sat, sun to sat, or *)
- (6) (the command to run, including options; also, this can be the name of a script with more commands)

Table 1: *at* Time Formats

Format	Meaning
<code>16:16</code>	16:16 hours today (or the next day, if it is past that time)
<code>07:00pm</code>	19:00 hours today (if you do not specify am or pm, am is assumed).
<code>now</code>	Right now
<code>tomorrow</code>	Tomorrow
<code>today</code>	Today
<code>now + 10min</code>	In 10 minutes time; you can also specify hours, days, weeks, and months.
<code>noon</code> <code>tomorrow</code>	At 12:00pm the next day; also, teatime (=4:00pm) or midnight.
<code>6/9/10</code>	June 9, 2010; or, for example, 6.9.10 and 6910.

If you want your computer to wake you at 7am every morning, enter:

```
0 7 * * * ogg123 -ZZ /home/huhn/music/*
```

The values in the individual fields can be separated by commas: To keep your alarm from ringing on Saturdays and Sundays, add this to the fifth weekday field:

```
0 7 * * 1,2,3,4,5 ogg123 -Zz /home/huhn/music/*
```

A combination of times can also be useful. You can specify a range with a dash (1-5), but weekday names are easier to read:

```
0 7 * 1-4,7,10-12 mon-fri ...
```

The values 1-4, 7, 10-12 in the fourth field (month) means "January to April, July, October to December." A slash followed by a number defines regular periods of time (e.g., */2 in the second column = "every two hours" and 1-6/2 = "1,3,5").

User cron tables are stored in the `/var` directory, but distros take different approaches when sorting the tables: Debian and Ubuntu store them in `/var/spool/cron/crontabs/` and sort by username; openSUSE uses `/var/spool/cron/tabs/`. As a normal user, you do not have read permission, but you can display your cron table by running the `crontab` program:

```
$ crontab -l
10 8 * * mon-fri ogg123 -Zz /home/huhn/music/*
```

To delete individual entries, launch the editor with `crontab -e`; if you intend to delete the whole table, run `crontab -r` instead.

INFO

[1] Browser-based Cronjob Monitor: <https://crontab.guru/>

Global cron Tables

Cron not only handles user-specific lists, it helps the root user with administration tasks. Working as root, look at the `/etc/crontab` file, which shows which jobs cron handles. Depending on the distribution, the global crontab can vary; Debian and Ubuntu have the entries shown in Listing 1.

Listing 1: Ubuntu and Debian crontab Entries

```
# m h dom mon dow user  command
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --
report /etc/cron.daily )
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --
report /etc/cron.weekly )
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --
report /etc/cron.monthly )
```

In contrast to normal user crontabs, the global crontab has a seventh field with the name of the user and the privileges for whom the command will run (typically root). This list tells you that the cron daemon runs `run-parts --report /etc/cron.hourly` with root privileges once an hour at 17 minutes past the hour, and at 6:52am on the first day of each month, cron runs `run-parts --report /etc/cron/monthly`. Cron takes care of the daily chores (the executable scripts in `/etc/cron.daily`) at 6:25am, including the `logrotate` script, which rotates, compresses, and sorts logfiles.

If you do not run your computer 24/7, modify these entries and specify times when you know your computer will be up:

```
25 17 * * * root
test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
```

Cron Alternatives

Several GUI-based tools will help you create a cron table. Gnome users have **Gnome Schedule** (package `gnome-schedule`), an easy-to-use program that lets you put together `at` and `cron` tasks with a few mouse clicks. The KDE tool is **KCron** (*System Settings | Startup and Shutdown | Task Scheduler.). KCron lets you modify the system-wide crontab, as well as cron and At schedules for certain user accounts.

In the end (and as in most cases), the command line gives you much more flexibility, and you can type entries much faster than if you were to click and point.

Alternatives such as **Anacron** and **Fcron** are available online or through your distro's package manager. Some of these tools provide enhanced scheduling features and even offer a way to "catch up" by executing tasks that were scheduled to run when the system was turned off.

Restrictive Bash

`rbash` or `bash -r`

exit restrictive bash with Ctrl + D

Documentation

The most basic form of help is the man page. Man pages are divided into eight sections, but most of the time, you only need to type the command **man** followed by the command, file, or concept about which you want information.

However, some topics have entries in several sections. To go to a specific section, place the number of the section between the **man** command and the topic. Thus, **man man** takes you to the basic page about the **man** command in section 1, but **man 7 man** takes you to a section about the collection of macros used to create man pages. Either way, when you are finished reading, you can press Ctrl+Z followed by Ctrl+C to return to the command line.

Man Page Sections	
Section	Description
1	General commands
2	System calls
3	C library functions
4	Special files (usually devices found in /dev) and drivers
5	File formats and conventions
6	Games and screensavers
7	Miscellanea
8	System administration commands and daemons

When you are doing deeper research, consider using **apropos** followed by a topic to receive a list of all the applicable man pages. **The one drawback to apropos is that, unless you are very specific, you could get dozens of pages, only a few of which are relevant to you.**

By contrast, if all you need is a brief snippet of information, use **whatis** followed by the command. For example, if you enter **whatis fdisk**, you receive the line **fdisk (8) -- Partition table manipulator for Linux**. The (8) refers to the man section where detailed information is available. Similarly, if you need to identify a file type, use **type**, then the file. Additionally, the GNU Project

made **info** its official help format. **But, instead of replacing man, info has simply become an alternative.**

Another important reference is the online Bash Reference Manual.

INFO

[1] Bash Reference Manual: <http://www.gnu.org/software/bash/manual/bashref.html>

Cal, date, hwclock, and NTP

An incorrectly set PC clock can be disastrous - if your computer loses track of the time, you could end up juggling files from the future or emails from 30 years ago. The time warp could lead to misunderstandings, errors, or even crashes.

Almost all Linux distributions set the time and time zones during the installation phase, and desktop environments such as KDE and Gnome display a clock in the panel to give users quick access to tools for configuring the computer clock.

In the shell, *cal* displays a simple but neatly formatted calendar. The *date* command gives you the date and time, although the output itself is fairly sparse. Additionally, this program can help the administrator set the date and time. The *date* tool also demonstrates its potential in combination with other command-line tools and in scripts, for example, when programs generate file names that contain the current date.

==The system time is maintained by the operating system, it is the one the processes will get when querying the date/time. Being stored in RAM, reading it is a fast operation.==

==The hardware time is maintained by a real clock powered by a battery. That means this clock persists a reboot. However, reading it implies performing a I/O operation which is more resource intensive than reading the system clock.==

==For that reason, the hardware clock is seldom used, mainly at boot time to set the system clock initial value, and then optionally to adjust/synchronize it either manually or through NTP.==

==Note that the hardware clock might be set to either the local time or UTC time while the system clock is always set on Unix/Linux systems to UTC time.==

==The *hwclock* tool helps to synchronize the system time and the hardware clock. Of course, you will need to be root to run this program.==

==If your machine has a permanent internet connection, you can automate the process of setting the clock by synchronizing your own timekeeper with a server on the web via the Network Time Protocol (NTP).==

Command-Line Calendar

If you call the calendar with *cal* and without parameters, the program displays the current month of the current year; the current day is highlighted. The *-3* option tells *cal* to show you the previous and next months as well; the *-y* flag produces a year calendar.

To output a specific month, you need to pass the month to *cal* in the form of a two-digit number for the month and a four-digit number for the year. By default, *cal* will output the calendar in the language defined in the *LANG* environment variable.

If you prefer the time format for any other language, but would like to keep output from all other programs in the default language, you can set the `LC_TIME` variable to tell `cal` to use the language of your choice.

The following example sets the date and time output to English:

```
LC_TIME=C cal -y
```

Of course, this command isn't necessary if your default language is already English.

What's the Time?

If you type `date` at the command line, you will see the date, time, and also the time zone:

```
$ date
Tue Feb 19 15:23:41 CST 2019
```

The `date` command also references the `LANG` variable to set the language, and it can also be influenced by setting `LC_TIME` just like `cal`.

`date` is even more flexible if you set the `TZ` (time zone) variable with the command:

```
$ TZ =America/New_York date
Tue Feb 19 16:27:24 EST 2019
```

If you happen to live in Australia and need to phone friends in New York on a regular basis, you might want to set up an alias for the last command to make it easier to check the time before you call.

To set up an alias, just add the following line to your Bash configuration file, `*~/.bashrc`

```
alias NY='TZ=America/New_York date'
```

and re-parse the settings after saving them by giving the **source** `~/.bashrc` command. Then, you can simply type `NY` at the command line to output New York time.

Formatted Output

The `date` program has a large number of parameters that influence the output format. You can format the `date` output with a plus sign, followed by a percent sign, and a letter. For example:

```
$ date +%Y_%m
2019_02
```

Table 1 lists some of the more common options; you can refer to the man page (`man date`) for a complete list of the options.

Table 1: Date Command-Line Parameters

Parameter	Meaning
%M	Minutes (00 to 59)
%H	Hours, 24-hour clock
%l	Hours, 12-hour clock
%a	Weekday, short form
%A	Weekday, long form
%d	Day as two-digit number
%b	Name of month, short form
%B	Name of month, long form
%m	Month as two-digit number
%y	Year as two-digit number
%Y	Year as four-digit number
%D	Four-digit date (mm/dd/yy)
%T	Time in 24-hour clock
%r	Time in 12-hour clock
%t	Tabulator
%n	Line break
%%	% sign

These formatting options are particularly practical if you use *date* to generate file names made up of date, time, or both values automatically.

The command

```
tar -cvjf backup_$(date +%d_%m_%Y).tar.bz2 *
```

creates a Bzip2 compressed tarball with a name comprising the text string *backup_*, the date (that is the day, month, and year separated by underlines), and the file extension **.tar.bz2* (for example, *backup_05_11_2009.tar.bz2*).

Setting the System Time

The root user can use *date* to set the time and date for a machine. To do so, use the *-s* option followed by a string that contains the new time (see the next section, "Everything is Relative"). Before you enter the following command, make sure that all NTP components have been uninstalled (see the "Automated" section):

```
date -s "19 Feb 2019 16:20"
date
Tue Feb 19 16:20:03 CST 2019
```

The first three parts of this are mandatory; if you leave out the year, *date* will just default to the current year.

Other format options let you set the date with seconds' precision. For this information, enter *man date* and study the date string examples.

Everything Is Relative

As an alternative to the absolute date and time, the *date* tool also understands relative values and even has a couple of predefined strings to help you:

- *yesterday*
- *tomorrow*
- *today*
- *now*
- *sec(s)/second(s)*
- *min(s)/minute(s)*
- *hour(s)*
- *day(s)*
- *week(s)*
- *fortnight*
- *month(s)*
- *year(s)*

Additionally, *date* understands concepts such as *ago*, so you can say *day ago* instead of *yesterday*.

```
date -d '+yesterday'
# or
date -d '-yesterday'
# ^^ both of the commands shown above display the same relative time.
# when using an option to specify date(s), any non-option
# argument must be a format string beginning with '+'
```

If you use one of these strings to set the time, you must specify the *-s* parameter like so:

```
date -s '+3 mins'
# ^^ sets the software clock 3 minutes forward in time
```

To display a relative time, you need the *-d* parameter instead:

```
date -d '+5 days -2 hours'
Mon Nov 29 11:02:47 AM CST 2021
# ^^ displays time relative to the current time (5 days in the future, 2 hours
prior to the current time)
```

The *date* information page tells you more about strings and how to use them. To read the documentation at the command line, use *info coreutils date*.

Setting the Hardware Clock

In addition to the software clock, your computer has another timekeeper, and this one will continue to count down the days when your computer is switched off and even when it is not plugged in.

To ensure uninterrupted timekeeping, computer mainboards have a battery-buffered clock, referred to as the CMOS clock, RTC (Real-Time Clock), BIOS clock, or even hardware clock.

The *hwclock* program lets you read and set the hardware clock; the commands all require root privileges. When used in combination with the *-r* option, you can display the local hardware time:

```
hwclock -r
2019-02-19 15:44:09.49830-0500
```

Additionally, *hwclock* has options for setting the system time to reflect the hardware clock time (*hwclock -s*) or vice versa (*hwclock -w*).

A combination of *--set* and *--date* sets a specific time. You need to enter a string to describe the new date and time after the *--date* parameter. The format is exactly the same as the date program's *-s* option. The command

```
hwclock --set --date="+2 hours"
```

sets the hardware clock to a time two hours in the future.

The Hardware Clock's Verbose Output

```
time sudo hwclock --verbose
```

```
$ time sudo hwclock --verbose
[sudo] password for cory:
hwclock from util-linux 2.36
System Time: 1639081161.332658
Trying to open: /dev/rtc0
Using the rtc interface to the clock.
Assuming hardware clock is kept in UTC time.
Waiting for clock tick...
...got clock tick
Time read from Hardware Clock: 2021/12/09 20:19:22
Hw clock time : 2021/12/09 20:19:22 = 1639081162 seconds since 1969
Time since last adjustment is 1639081162 seconds
Calculated Hardware Clock drift is 0.000000 seconds
2021-12-09 14:19:21.343237-06:00

real    0m5.062s
user    0m0.026s
sys     0m0.000s
```

INFO

[1] <http://www.unixtimestampconverter.com/>

[2] https://labs.isee.biz/index.php/How_to_adjust_Hardware_RTC_clock

[3] <http://manpages.ubuntu.com/manpages/jammy/en/man8/ntpd.8.html>

[4] <https://titanwolf.org/Network/Articles/Article?AID=1adcffb8-e870-4fe4-ba43-8d0a65f64777>

<https://support.apple.com/guide/shortcuts/technical-standards-apdfd459e13d/ios>

<https://datatracker.ietf.org/doc/html/rfc2822#section-4>

<https://en.wikipedia.org/wiki/UTC%E2%88%9206:00#/media/File:Timezones2008 UTC-6 gray.png>

<https://greenwichmeantime.com/what-is-gmt/>

Automated!

Network Time Protocol (NTP) is a standard for automating the synchronization of clocks in computer systems. The time signal propagates over the network from an NTP server to a client, and you can configure the point in time when your Linux machine's NTP client contacts a server on the network. This could be at boot time or when you get onto the Internet, or you could use a manual command in the shell.

In the pre-Systemd era, most major Linux distributions had packages available for enabling NTP support. Many of those packages still exist - see the documentation for your own Linux distribution to learn about NTP package options.

Systemd provides a built-in *systemd-timesyncd* services that performs basic time synchronization duties. To check whether the service is running on your system, enter:

```
systemctl status systemd-timesyncd.service
```

The *systemd-timesyncd* service is like other Systemd services. You can start, stop, or restart it using a variation of the *systemctl* command:

```
systemctl restart systemd-timesyncd.service
```

See the article on Systemd elsewhere in this issue, or consult the *systemctl* man page, for more on managing Systemd services.

INFO

[1] NTP: (http://en.wikipedia.org/wiki/Network_Time_Protocol)

[2] NTP: (<http://manpages.ubuntu.com/manpages/jammy/en/man8/ntpd.8.html>)

Secure connections with SSH

Telnet is probably the best-known solution for providing users with console access to remote machines. However convenient this dinosaur of network communication might be, it has one major disadvantage: All the data is sent in plaintext over the wire. If an attacker sniffs the connection, he or she will quickly learn the administrative password for the server. Admittedly, it probably isn't quite that easy, but the danger is there all the same. For this reason, all popular Linux distributions install the Secure Shell (SSH) as a safer alternative.

Getting Started with SSH

First, a bit of theory. The SSH client/server architecture is based on TCP/IP. The SSH server (`sshd`) runs on one machine, where it listens for incoming connections on TCP port 22. The client simply uses this port to connect to the server. When a connection is established, several things happen in the background. The server and client exchange information about supported protocol versions to use for communications. Currently SSH1 and SSH2 are available, but SSH2 is standard today because of its better security. Details - including details of encryption - are given in the "SSH Protocol Versions" box. The server and client then negotiate the algorithm, followed by the key that both will use for the data transfer. The key is used only once for the current communication session, and both ends destroy it when the connection is broken. For extended sessions, the key will change at regular intervals, with one hour being the default.

To get started with SSH, you need to install an SSH server on the target machine - and you can't go wrong with OpenSSH, which is by far the most popular SSH software on Linux. To install the OpenSSH server on Debian and Ubuntu, run the

```
apt-get install openssh-server
```

command as root.

Once the installation is completed, the server is ready to go. Although it runs perfectly well with the default configuration, you might want to change the default port 22 the server is running on. Open `sshd_config` for editing using your favorite editor as root and change the `Port` value (e.g., `Port 1777`). Then, restart the SSH server using the

```
/etc/init.d/ssh restart
```

command as root. This simple trick makes the lives of potential intruders slightly more difficult, because many malicious port scanners check the default port 22 and move on if it's not open. Of course, this doesn't make your server completely secure, but every bit helps.

To connect to the server via SSH, you also need to install the OpenSSH client on your machine using `apt-get install openssh-client` as root. To establish an SSH connection, open the terminal and run the `ssh user@remotehost` command (replace USER with the actual user name on the server and HOST with the IP address or domain name of the server). If you changed the default SSH port, then you need to specify the port parameter explicitly:

```
ssh -p 1777 user@remotehost
```

On first log in, the client will not know the server's host key and will prompt you to confirm that you really do want to establish a connection with the remote machine. After confirming, the program generates the fingerprint (Figure 1).

Figure 1: On initial login, SSH imports the host key.

```

debian:~# ssh sector
The authenticity of host 'sector (192.168.10.100)' can't be established.
RSA key fingerprint is 81:00:6e:dc:49:e1:5b:1d:76:86:8c:a4:55:91:0d:29.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added 'sector,192.168.10.100' (RSA) to the list of known
hosts.
Password:
Last login: Tue Sep 27 14:45:53 2005 from 192.168.10.254
Loading /usr/share/keymaps/i386/qwertz/de- latin1-nodeadkeys.kmap.gz
SECTOR:~$

```

SSH Protocol Versions

SSH1 and SSH2 are the current versions. SSH1 uses the insecure DES or the secure Triple DES (3DES). The Blowfish algorithm provides a fast and - so far - secure encryption technology. Version 2 includes the AES algorithm and others. Vulnerabilities in the SSH1 protocol make it possible to hack the encryption. Version 1 relies on encryption of data with a random number that has been encrypted with the server's public key. This method is open to brute force attacks that give the attacker the plaintext key. Protocol 2 relies on a Diffie-Hellman exchange that never transmits the key over the wire but allows server and client to generate the same key independently. Other enhancements to version 2 include the software's ability to check the data integrity with cryptographic hashes (the Message Authentication Code method) rather than the unreliable CRC (Cyclic Redundancy Check) method. Support for multiplexing is also improved. All of the examples in this article use SSH2, although some will work with SSH1.

To check the key fingerprint, contact the administrator of the remote machine. This prevents man-in-the-middle attacks, in which an attacker reroutes network traffic to his own machine while spoofing a genuine login to your machine. If you were to confirm the security prompt and enter your password, the attacker would then own your password; thus, some caution is recommended. If the host key changes, the client will refuse to connect when you log in later. Figure 2 shows the output from the SSH client.

Figure 2: If the host key changes, the SSH client will refuse to connect.

```

debian:~# ssh sector
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@   WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!   @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
81:00:6e:dc:49:e1:5b:1d:76:86:8c:a4:55:91:0d:29.
Please contact your system administrator.
Add correct host key in /root/.ssh/known_hosts to get rid of this message.
Offending key in /root/.ssh/known_hosts:1
RSA host key for sector has changed and you have requested strict checking.
Host key verification failed.
debian:~#

```


The only thing that will help is to remove the offending fingerprint from your `$HOME/.ssh/known_hosts` file and accept the new key after contacting the administrator on the remote machine. To configure this behavior, use the `StrictHostKeyChecking` variable in `ssh_config`.

If you do not want to use your current account name to log in to the remote machine, the `-l` `login_name` option can help. For example, the command

```
ssh -l tuppes sector
```

logs you in to the remote machine as user `tuppes`. SSH also accepts the syntax `ssh tuppes@sector`. To run a single command on the remote machine, you simply append it to the command line (Listing1).

Listing 1: Running Commands on the Remote Machine

```
jha@scotti:~$ ssh sector "ls -l"
Password:
insgesamt 52
Drwxr-xr-x 3 tuppes users 4096 2005-08-26 12:38 .
Drwxr-xr-x 16 root root 4096 2005-09-07 13:47 ..
-rw-rw-r-- 1 tuppes users 266 2005-04-12 12:00 .alias
```

If you get tired of typing your password, public key authentication provides an alternative. This technique uses encryption methods such as those used by GnuPG. Before you can use the public key approach, you first need to run `ssh-keygen` to generate a pair of keys:

```
ssh-keygen -b 2048 -t rsa
```

The software will tell you that it has created a keypair with a public key and private key on the basis of the RSA approach. When prompted to enter a password, press `Enter` twice. The program will then tell you where it has stored the data and will display the fingerprint for the new key.

In the example here, the software generates an RSA keypair (`-t rsa`) with a length of 2048 bits (`-b 2048`). For security reasons, the key length should not be less than 2048 bits. **To be absolutely safe, you can use a key length of 4096 bits.** The key length has no influence on the data transfer speed because the program does not use this key to encrypt the data.

Next, copy the public key to the `$HOME/.ssh/authorized_keys` on the remote machine from, for example, a floppy disk:

```
mount /media/floppy
cat /media/floppy/id.rsa.pub >> $HOME/.ssh/authorized_keys
umount /media/floppy
```

Certainly you should avoid transferring the key by an insecure method, such as email or FTP. Figure 3 shows the fairly unspectacular login with the new key.

Figure 3: Public key authentication makes the login more user friendly by removing the password prompt.

```
debian:~# ssh sector
Last login: Wed Sep 28 13:36:22 2005 from 192.168.10.254
SECTOR:~$
```

Passwords protect keys for interactive sessions; otherwise anybody with physical access to your computer could use your keys to log in to the remote machine. Key-based, password-free logins are often used to automate copying of files to remote machines.

For example, if you back up your data every evening and want to copy it automatically to a remote machine, keys without passwords are a useful approach. If the key was password protected, you would need to enter the password for the SSH key to copy the data - so much for automated copying.

Typing SSH commands like

```
ssh -p 1777 pi@192.168.101
```

can become a nuisance if you have to do this several times a day. Fortunately, you can solve this problem by defining SSH aliases for often-used SSH connections in the `~/.ssh/config` file:

```
Host alias
  HostName ipaddress
  User username
  Port portnumber
```

Replace `alias` with the desired alias name, `ipaddress` with the IP address or domain name of the server, `username` with the actual username, and `portnumber` with the appropriate port number:

```
Host raspberrypi
  HostName 192.168.1.101
  User pi
  Port 1777
```

You need to specify the `Port` parameter only if the SSH server is running on any port other than 22.

Useful Freebies

The SSH package includes two more useful programs: Secure Copy (`scp`) and Secure FTP (`sftp`). As the names suggest, these programs are used to copy and transfer files by FTP via SSH. The basic syntax for the two programs is similar.

For example, to copy `test.txt` from your home directory on the remote machine to your current working directory, enter:

```
scp RemoteComputer:test.txt .
```

Depending on your authentication method, you might need to enter your password; however, the colon is mandatory in all cases. It separates the name of the remote machine from the pathname. Also, you need to specify the local path. The easiest case is your current working directory, which is represented by the dot at the end of the line. To copy multiple files, just type a blank-delimited list of the file names:

```
scp RemoteComputerA:test1.txt RemoteComputerB:test2.txt .
```

If you use the standard login approach, the client will prompt you to enter your password for each file you copy. If you use the public key method discussed previously, you don't need to type a password. The command

```
scp RemoteComputerA:test.txt RemoteComputerB:
```

copies the file from remote computer A to remote computer B. To copy a file as *tuppes* from `/home/tuppes/files` to your local directory, type:

```
scp tuppes@RemoteComputer:files/test.txt .
```

Unlike SSH, you do not specify the `-l username` option. Copying in the other direction - local to remote - is just as easy:

```
scp ./test.txt tuppes@RemoteComputer:/files/
```

`scp` copies the `test.txt` from your current working directory to `/home/tuppes/files` on the remote machine. Again, watch out for the closing colon. The `sftp` tool uses the same command structure as `scp` but has two operating modes: interactive, like the one you might be familiar with from FTP, and batch mode. To use `sftp` to retrieve the sample file from the remote machine in batch mode, type:

```
sftp RemoteComputer:test.txt .
```

If you add `remote_test.txt` to the end, the program will give that name the local copy of the file. Typing `sftp RemoteComputer` opens an interactive, encrypted FTP session on the remote machine, and the server will accept FTP commands such as GET or PUT in the session.

Building Tunnels

SSH also lets you encapsulate other protocols. For example, you can run the Telnet protocol over an encrypted SSH connection and do it transparently for users. The technical term for encapsulating one protocol inside another is "tunneling."

The standard specifies that programs must be running on the same machine to use the tunnel. If you want other machines on the network to use the tunnel, you must specify `-o GatewayPorts=yes` when setting up the tunnel. The alternative is to set the option in `ssh_config`.

This setup is similar to a Virtual Private Network (VPN) connection but is easier to implement. The SSH variant has the disadvantage that you can only forward a single TCP port. Thus, you need an SSH tunnel for each port you want to forward. If you want to encrypt all communication between two machines, a VPN is probably a better choice.

Any user can set up a tunnel, although tunnels for privileged ports (i.e., below 1024) are reserved for root. To open a tunnel to a remote machine encapsulating the Telnet protocol (port 23), enter:

```
ssh -c blowfish -L 23 RemoteBox:23RemoteBox
```

The command uses the `-L` option to open a tunnel from local port 23 on the local machine (the first 23) to port 23 on the remote machine. The fast Blowfish method is used for encryption. If you type two remote machine names, you can take advantage of another SSH feature: opening a tunnel from the first machine, via the second, to a third. The command

```
ssh -L 23:192.168.1.1:23 192.168.20.5
```

starts the tunnel on the local machine and routes it via an intermediate station (192.168.1.1) to its endpoint. The generic syntax for opening a tunnel from the local machine to the remote computer is thus:

```
ssh -L LocalPort:RemoteComputerA:RemotePort RemoteComputerB
```

For a direct tunnel, the two host designations are identical.

Tunnel Tricks

Netstat is considered obsolete with introduction of systemd, even though it is still functional on most distros. Replacements for the `netstat` commands are `ss` (socket statistics), which takes many of the same options as `netstat`, and `ip`, which replaces `ifconfig` (see the Networking Tools article). Table 1 shows the commands that replace common `netstat` commands.

Table 1: netstat Replacement Commands

netstat	Replacement
<code>netstat</code>	<code>ss</code>
<code>netstat -r</code>	<code>ip route</code>
<code>netstat -i</code>	<code>ip -s link</code>
<code>netstat -g</code>	<code>ip maddr</code>

```
[Fri May 12 09:30 PM]$sudo netstat -tlnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
    PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
    501/sshd
tcp        0      0 127.0.0.1:6010          0.0.0.0:*               LISTEN
    868/0
tcp6       0      0 :::22                  :::*                    LISTEN
    501/sshd
tcp6       0      0 :::1:6010              :::*                    LISTEN
    868/0
[Fri May 12 09:33 PM]$
[Fri May 12 09:33 PM]$sudo ss -tlnp | cat
```

```

State      Recv-Q  Send-Q      Local Address:Port      Peer Address:Port
LISTEN    0        128          *:22                     *:
users: (("ssh", pid=501, fd=3))
LISTEN    0        128          127.0.0.1:6010          *:
users: (("ssh", pid=868, fd=11))
LISTEN    0        128          :::22                    :::
users: (("ssh", pid=501, fd=4))
LISTEN    0        128          :::1:6010                :::
users: (("ssh", pid=868, fd=10))
[Fri May 12 09:33 PM]$
[Fri May 12 09:33 PM]$sudo netstat -tbn
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
      PID/Program name
tcp    0      0 192.168.2.4:22          192.168.2.1:58712      ESTABLISHED
      862/ssh: pi [priv]
[Fri May 12 09:34 PM]$
[Fri May 12 09:34 PM]$sudo ss -tbn | cat
State      Recv-Q  Send-Q      Local Address:Port      Peer Address:Port
ESTAB     0        0          192.168.2.4:22          192.168.2.1:58712
users: (("ssh", pid=868, fd=3), ("ssh", pid=862, fd=3))

```

In Figure 4, the `netstat` and `ss` commands demonstrate that a Telnet connection has been set up via SSH. The first set of commands (`netstat` and `ss`) shows that an SSH process with an ID of 501 is listening on port 22. The second set of commands show that a connection to port 22 is established.

If you were to look more closely at the syntax used to open a tunnel, you might be led to assume that the local and remote ports do not need to be identical - and this is true. Assuming the remote machine is running a proxy configured for transparent proxying on port 3128, you could redirect all HTTP requests as follows:

```
ssh -o GatewayPorts=yes -L 80:RemoteComputer:3128RemoteComputer
```

This process of redirecting one port to another is called **port forwarding**. Other computers on the network also need `-o GatewayPorts=yes` to use the tunnel.

Similarly, tunneling works in the reverse direction. The following syntax

```
ssh -o GatewayPorts=yes -R 3128:LocalComputer:80RemoteComputer
```

lets you set up a return tunnel from the remote machine to your computer.

Graphical Tunnels

The X Window System is natively network-capable, but almost nobody uses this ability because communications are again unencrypted over the wire. Tunneling with SSH makes this a far more attractive proposal.

To tunnel X11, the SSH daemon (`sshd`) emulates an X server and occupies a display (number 11 by default). When you log in to the server, the server sets the `DISPLAY` environment variable to this value, or to `localhost:11.0` to be more precise. The idea is to avoid collisions with the X server running locally. Information sent by a computer to this display is encrypted and sent to your machine.

OpenSUSE 11.0 enables X11 forwarding (the technical term for the process) by default. You can disable X11 forwarding on the machine configured for forwarding by setting the `X11Forwarding` variable to `no` in `etc/ssh/sshd_config`. The `X11DisplayOffset` variable with a default value of `10` defines the distance between the virtual display and the physical display; you should keep the default here.

If the machine on which you want to display tunneled X11 is an OpenSUSE 11.0 machine, the `etc/ssh/ssh_config` file will already have the `ForwardX11Trusted` variable set to `yes`.

Next, log in to the remote machine and launch, for example, the `Xclock` program. Figure 5 (Linux Shell Handbook - 2021 Edition, pg. 75) shows the display (`localhost:11.0`), the process, and the matching network connections.

Other Practical Uses

SSH can be put to other clever uses to. For example, the popular backup tool `rsync` is perfectly capable of copying files via SSH. Thus, you can easily implement an offsite backup solution via `rsync` and SSH, for which you need to install OpenSSH and `rsync` on both the source and destination machines. To perform an offsite backup, specify the path on the remote host as the destination:

```
rsync -avzh /path/to/local/dir user@remotehost:/path/to/remote/dir
```

Usually, you don't have to craft any special command to use `rsync` via SSH. However, if you are using a non-standard port or a non-default private key location, you'll need to use the `-e` switch:

```
rsync -avzhe 'ssh -p 1777' /path/to/local/dir user@remotehost:/path/to/remote/dir
```

As described earlier, you can use `scp` to copy files securely via SSH. For more efficient compression and transfer, however, you might want to use a simple trick that allows you to copy multiple files and directories between local and remote machines by piping a `tar` datastream to SSH. Simply put, you can use the `tar` tool to compress several files or a directory and send the resulting archive to a remote host via SSH:

```
tar zcf - DIR | ssh user@remotehost 'tar zxf -'
```

This command compresses the contents on the `DIR` directory, passes the resulting archive to SSH, and then decompresses the archive on the remote host. Of course, you can perform the same action in the opposite direction, from the remote host to the local machine:

```
ssh user@remotehost 'tar zcf - DIR' | tar zxf -
```

Using SSH, you can also mount a remote filesystem on your local machine and manage files and documents stored on a remote host as if they were stored locally. To enable this functionality, you need to install the `sshfs` package. On Debian and Ubuntu, this can be done by running the `apt-get install sshfs` command as root. You can then use the `sshfs` command to mount a remote directory. The following example mounts the `remotedir` directory in the `/home/user/dir` path on the local machine:

```
sshfs user@remotehost:  
/path/to/remotedir /home/user/dir -o idmap=user -o uid=1000 -o gid=1000
```

The `-o` parameters make the mounted directory writable by the local user with the specified user and group IDs.

Sometimes, you might need to run certain commands on the remote host as soon as you establish an SSH connection:

```
ssh -p 1973 pi@192.168.1.101 -t "cd /path/to/dir ; ./foo.sh ; /bin/bash"
```

As soon as an SSH connection is established, the command switches to the specified directory and runs the `foo.sh` script in the Bash shell.

Conclusions

The SSH package includes a collection of important programs that make working on networks far more secure. The feature scope covers anything from basic encrypted connections, through tunneling and port forwarding, to X11 forwarding.

Figure 5: A forwarded X11 connection - Xclock is running on a remote machine.

```
harmuth@debian:~$ ssh sector
Last login: Wed Sep 28 16:42:42 2005 from 192.168.10.254 on ttyp3
Linux SECTOR 2.4.31 #1 SMP Do Aug 4 14:24:58 CEST 2005 i686 unknown
You have mail.

Last login: Wed Sep 28 16:42:52 2005 from 192.168.10.254
harmuth@SECTOR:~$ xclock &
[1] 8056
harmuth@SECTOR:~$ echo $DISPLAY
localhost:11.0
harmuth@SECTOR:~$ ps aux | grep xclock | grep -v grep
harmuth  8056  1.0  0.3  5280  3308  ttyp3    S   16:43   0:00  xclock
harmuth@SECTOR:~$ su -
Password:
SECTOR:~$ netstat -tbn | grep xclock
tcp        0      0  127.0.0.1:52503      127.0.0.1:6011      VERBUNDEN
8056/xclock
SECTOR:~$
```

Synchronizing Data with Rsync

Rsync is the perfect synchronization tool for keeping your data in sync. The program manages file properties and uses SSH to encrypt your data, and it is perfect for transferring large volumes of data, if the target computer has a copy of a previous version.

Rsync checks for differences between the source and target versions. The tool that has been developed by the Samba team uses an efficient checksum-search algorithm for comparing data; rsync only transfers the differences between the two sides and therefore saves time and bandwidth.

In Sync

The generic syntax for rsync is `rsync [options] source target`, where `target` can be a local target on the same machine or a remote target on another machine. The choice of source and target is critical; decide carefully in which direction you will be synchronizing to avoid loss of data. If you're not sure that you are using the correct options or the correct source/target, you can run rsync with the `-n` flag to tell the program to perform a trial run. Additionally, you can increase the amount of information by defining `-v` and switching to verbose output.

To mirror a directory `dir1` on a local machine, for example, type:

```
$ rsync dir1/* dir2/
skipping directory foo
skipping directory bar
skipping non-regular file "text.txt"
```

As the output shows, rsync would transfer normal files but leave out subdirectories and symbolic links (`non-regular file`). To transfer directories recursively down to the lowest level, you should specify the `-r` option. Using the `-l` flag additionally picks up your symlinks. Of course, a combination of the options is also possible:

```
rsync -lr dir1/* dir2/
```

Rsync has an alternative approach to handling symlinks. If you replace `-l` with `-L`, the program will resolve the link, and your former symlinks will end up as "normal" files at the target.

Be careful with the slash - appending a slash to a directory name influences the way rsync handles an operation (see the "Common Rsync Traps" box).

As You Were

If you will be using rsync to create backups, it makes sense to keep the attributes of the original files. By attributes I mean permissions and timestamps - that is, information on the last access time (`atime`), the last status change (`ctime`), and the last modification (`mtime`).

Additionally, administrators can benefit from parameters that preserve owner and group data and support device files. To retain the permissions, just specify the `-p` option; `-t` handles the timestamps, and `-g` keeps the group membership.

Whereas any normal user can specify these parameters, the `-o` (keep the owner data) and `-D` (device attributes) flags are available only to root. The complete command line with all these options could look like this:

```
rsync -rLptgoD /home/huhn /backup
```

Don't worry - you don't have to remember all these options. Rsync offers a practical shortcut and a special option that combines these parameters for this case. Instead of `-rLptgoD`, just type `-a`.

Exclusive

Rsync has another practical option that allows you to exclude certain files from the synchronization process. To leverage this feature, specify the `--exclude=` option and a search pattern and define the files to exclude. With this option, you can use wildcards:

```
rsync -a --exclude=*.wav ~/music backup/
```


This example excludes large WAV files that end in `.wav` from the backup of a music collection. If you need to exclude MP3s as well, just append another `exclude` statement and a pattern:

```
rsync -a --exclude=*.wav --exclude=*.mp3 ...
```

To save time, you can store your exclusions in a text file. To do this, you will need a separate line for each search pattern. Specify the `--exclude-from=file_with_exclusions` parameter to parse the file.

Tidying Up

Rsync offers various parameters for deleting data that is no longer needed or wanted. To get rid of files in your backup that no longer exist in the source, type `--delete`. Rsync's default behavior is to delete files before the transfer is finished. Alternatively, you can define `delete-after` to delete files of the target after all the syncing is done.

Additionally, you can tell rsync to delete files that you have excluded. For example, imagine you've decided that you no longer want the MP3s in the backup and you've started to exclude them with `--exclude=*.mp3`. Now you can define `--delete-excluded`, and rsync will recognize that those files are no longer wanted.

All `--delete` options have basically the same goal: to keep an exact copy of the original. If you don't use the switch, you will have to clean up manually; otherwise, the files that you've decided are useless will remain. Use these options with care.

Common Rsync Traps

Some rsync options could cause trouble if you don't use them with caution. Being aware of these common mistakes can help.

- Most users find the final slash for directories confusing at first. For example, if you call `rsync -a source/folder target/`, rsync will transfer the directory called `folder` and its contents to the `target` directory. If the directory `folder` doesn't exist, rsync will create it. If you append a slash to `source/folder`, rsync will only transfer the contents of `folder`. That means a file `source/folder/foo.txt` is being transferred to `target/foo.txt` instead of `target/folder/foo.txt`.
- An absolute classic troublemaker is the option `--delete`. If you get source and target mixed up, `--delete` will happily delete several original files. To be on the safe side, remember to use `-n` in a test run.
- If a transfer is interrupted and you're using the `--partial` flag, rsync saves parts of the file under the same name as the original, which is not always helpful. Imagine that you're using rsync to update a large and existing ISO image of your favorite distribution (like a Release Candidate). The transfer of the new version gets interrupted after just a few bytes. Rsync will overwrite your original file with the smaller part of the ISO image from the server, and you will have lost your current file and have to start from scratch. To avoid data loss in this scenario, you can create a hard link before calling rsync. If the transfer fails now, you won't lose the ISO image; instead the partial file will be given a new name without destroying the original.

Tuning Rsync

Several options increase rsync's performance. Often, I use the `-z` switch to compress data when I sync data over a network connection. Figure 1 shows this using Grsync, the graphical front end to rsync. If the connection is very slow, you can also define a bandwidth limit. To transfer data with only 20KBps, for example, use:

```
rsync ... --bwlimit=20
```

Rsync is perfect for transferring large volumes of data. If you specify the `--partial` parameter and the transfer is interrupted for some reason, you can pick up the transfer from the point at which you left off. Specifying the `--progress` option gives you a progress indicator to let you keep track of the transfer operation:

```
$ rsync -avz --progress --partial remote.server:/home/huhn/music/folk ~/music/  
receiving file list ...  
42 files to consider  
...  
12_Moladh_Uibhist.mp3  
 1143849  4% 339.84kB/s  0:01:10
```

At the other end of the connection, the partial file is hidden in the target directory at first. Typing `ls -a` reveals a file called `.12_Moladh_Uibhist.mp3.7rUSSq`. The dot at the start of the file name keeps the file hidden, and the arbitrary extension removes the danger of overwriting existing files.

When the transfer completes, the file gets its original name back. If the transfer is interrupted, you can restart by specifying the `--partial` option again. Alternatively, you have a shortcut: If you want to use a combination of `--partial` and `--progress`, simply use `-P`.

Rsync keeps your data up to date and you stay on top of confusing version changes. Its options help you manage file properties, and it works well with SSH. When you need to transfer large volumes of data, rsync comes to your rescue.

INFO

[1] Rsync website: (<https://rsync.samba.org>)

[2] Grsync: (<http://www.opbyte.it/grsync/>)

Regular Expressions

Anchors

Characters	Meaning
^	Start of string, or start of line in multi-line pattern
\A	Start of string
\$	End of string, or end of line in multi-line pattern
\Z	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

Character Classes

Characters	Meaning
\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\x	Hexadecimal digit
\O	Octal digit

The POSIX Character Set table below has been taken from the grep info page with a tiny amount of editing, see [10] in the Bibliography for further information.

POSIX Character Sets

Characters	Meaning
[:upper:]	uppercase letters
[:lower:]	lowercase letters
[:alpha:]	alphabetic (letters) meaning upper+lower (both uppercase and lowercase letters)
[:digit:]	numbers in decimal, 0 to 9
[:alnum:]	alphanumeric meaning alpha+digits (any uppercase or lowercase letters or any decimal digits)
[:word:]	Digits, letters, and underscores
[:blank:]	Space and tab
[:space:]	whitespace meaning spaces, tabs, newlines and similar
[:graph:]	graphically printable characters excluding space
[:print:]	printable characters including space
[:punct:]	punctuation characters meaning graphical characters minus alpha and digits
[:cntrl:]	control characters meaning non-printable characters
[:xdigit:]	characters that are hexadecimal digits

Assertions

Characters	Meaning
?=	Lookaround assertion
?!	Negative Lookaround
?<=	Lookbehind assertion
?!= or ?<!	Negative Lookbehind
?>	Once-only Subexpression
?()	Condition [if then]
?()	Condition [if then else]
?#	Comment

Quantifiers

Characters	Meaning	Characters	Meaning
*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4, or 5

Add a **?** to a quantifier to make it ungreedy or lazy, matches as few characters as possible. By contrast, something that is greedy matches as many characters as possible.

Escape Sequences

Characters	Meaning
\	Escape following character
\Q	Begin literal sequence
\E	End literal sequence

"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.

Common Metacharacters

Characters	Meaning
^ (Caret)	means "the beginning of the line". So "^a" means find a line starting with an "a".
[] (Square Brackets)	specifies a range. If you did m[a,o,u]m it can become: mam, mum, mom if you did: m[a-d]m it can become anything that starts and ends with m and has any character a to d inbetween. For example, these would work: mam, mbm, mcm, mdm. This kind of wildcard specifies an "or" relationship (you only need one to match).
[!]	This construct is similar to the [] construct, except rather than matching any characters inside the brackets, it'll match any character, as long as it is not listed between the [and]. This is a logical NOT. For example rm myfile[!9] will remove all myfiles* (ie. myfiles1, myfiles2 etc) but won't remove a file with the number 9 anywhere within it's name.
[^]	This is the equivalent of [!] in standard wildcards. This performs a logical "not". This will match anything that is not listed within those square brackets. For example, <code>rm myfile[^9]</code> will remove all myfiles* (ie. myfiles1, myfiles2 etc) but won't remove a file with the number 9 anywhere within it's name.
{ } (Curly braces)	terms are separated by commas and each term must be the name of something, or a wildcard. This wildcard will copy anything that matches either wildcard(s), or exact name(s) (an "or" relationship, one or the other). For example <code>cp {*.doc, *.pdf} ~</code> will copy anything ending with .doc or .pdf to the users home directory. Note that spaces are not allowed after the commas (or anywhere else).
() Parentheses	Used for capturing groups.
\	The escape character is usually \ , i.e. to protect a subsequent special character. Thus, "\\ " searches for a backslash. Note you may need to use quotation marks and backslash(es).
.	will match any single character, equivalent to ? (question mark) in standard wildcard expressions. Thus, "m.a" matches "mpa" and "mea" but not "ma" or "mppa".
.*	is used to match any string, equivalent to * in standard wildcards.
*	this item is to be matched zero or more times. ie. n* will match n, nn, nnnn, nnnnnn but not na or any other character.
\$	means "the end of the line". So "a\$" means find a line ending with an "a". For example, this command searches the file myfile for lines starting with an "s" and ending with an "n". <code>cat myfile grep '^s.*n\$'</code>
	This wildcard makes a logical OR relationship between wildcards. This way you can search for something or something else (possibly using two different regular expressions). You may need to add a '\ ' (backslash) before this command to work, because the shell may attempt to interpret this as a pipe.

Characters	Meaning
?	Match 0 or 1 of the preceding token
+	Match 1 or more of the preceding token
<	Matches a "<" character
>	Matches a ">" character

Special characters

Characters	Meaning
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed
\xxx	Octal character
\xhh	Hex character hh

Groups and Ranges

Characters	Meaning
.	Any character except new line (\n)
(a b)	a or b
(...)	Group
(?:...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"

Ranges are inclusive.

Pattern Modifiers

Characters	Meaning
g	Global match
i *	Case-insensitive
m *	Multiple lines
s *	Treat string as single line
x *	Allow comments and whitespace in pattern
e *	Evaluate replacement
U *	Ungreedy pattern

* Perl-Compatible Regular Expressions (PCRE) modifier

String Replacement

Characters	Meaning
\$n	nth non-passive group
\$2	"xyz" in /^(abc(xyz))\$/
\$1	"xyz" in /^(?:abc)(xyz)\$/
\$`	Before matched string
\$'	After matched string
\$+	Last matched string
\$&	Entire matched string

Some regex implementations use \ instead of \$.

++Standard Wildcards (globbing patterns)++

Standard wildcards (also known as globbing patterns) are used by various command-line utilities to work with multiple files. For more information on standard wildcards (globbing patterns) refer to the manual page by typing:

```
man 7 glob
```

Can be used by: Standard wildcards are used by nearly any command (including mv, cp, rm and many others).

- ? (question mark)
: this can represent any single character. If you specified something at the command line like "hd?" GNU/Linux would look for hda, hdb, hdc and every other letter/number between a-z, 0-9.

- * (asterisk)
: this can represent any number of characters (including zero, in other words, zero or more characters). If you specified a "cd" it would use "cda", "cdrom", "cdrecord" and anything that starts with "cd" also including "cd" itself. "ml" could be mill, mull, ml, and anything that starts with an m and ends with an l.
- [] (square brackets)
:
- { } (curly brackets)
:: For example, this would be valid:

```
cp {*.doc, *.pdf} ~
```

: This

- [!]
:
- \ (backslash)
: is used as an "escape" character, i.e. to protect a subsequent special character. Thus, "\ " searches for a backslash. Note you may need to use quotation marks and backslash(es).

++Regular Expressions++

Regular expressions are a type of globbing pattern used when working with text. They are used for any form of manipulation of multiple parts of text and by various programming languages that work with text. For more information on regular expressions refer to the manual page or try an online tutorial, for example IBM Developerworks using regular expressions. For the manual page type:

Type:

```
man 7 regex
```

Regular expressions can be used by: Regular Expressions are used by grep (and can be used) by find and many other programs.

Tip: If your regular expressions don't seem to be working then you probably need to use single quotation marks over the sentence and then use backslashes on every single special character.

1. . (dot)
:
2. \ (backslash)
: is used as an "escape" character .* (dot and asterisk)
:
3. * (asterisk)
:
4. ^ (caret)
:
5. \$ (dollar sign)
:

```
::  
  bash  
  cat myfile | grep '^s.*n$'
```

6. [] (square brackets)
: specifies a range. If you did `m[a,o,u]m` it can become: `mam`, `mum`, `mom` if you did: `m[a-d]m` it can become anything that starts and ends with `m` and has any character `a` to `d` inbetween. For example, these would work: `mam`, `mbm`, `mcm`, `mdm`. This kind of wildcard specifies an “or” relationship (you only need one to match).
7. |
:
8. [^]
:

POSIX Character Sets

These are used with: The above commands will work with most tools which work with text (for example: `tr`).

For example (advanced example), this command scans the output of the `dir` command, and prints lines containing a capital letter followed by a digit:

```
ls -l | grep '[:upper:][:digit:]'
```

The command greps for `[upper_case_letter][any_digit]`, meaning any uppercase letter followed by any digit. If you remove the [] (square brackets) in the middle it would look for an uppercase letter or a digit, because it would become `[upper_case_letter any_digit]`

Hardware Help

In modern Linux, you can work with hardware in several different ways: general information commands, the `dmidecode` command, viewing the `/proc` and `/sys` virtual filesystems, manipulating kernel modules, modifying daemon configuration files, and working with `systemd`.

Table 1: Hardware Information Commands
<i>lsblk</i> Lists information about all available or specified block devices.
<i>lscpu</i> Summarizes CPU architecture information.
<i>lshw</i> Displays hardware information.
<i>lspci</i> Extracts detailed information on the machine's hardware configuration.
<i>lsusb</i> Displays information about USB buses in the system and the devices connected to it.
<i>uname</i> Displays software and hardware information.

Important Lshw Options

Output and Display
<i>-html</i> Generate HTML output
<i>-xml</i> Generate XML output
<i>-short</i> Show a short summary
<i>-businfo</i> Output bus information
<i>-X</i> Use the graphical interface
Actions
<i>-c, -C, -class</i> Show class information.
<i>-disable</i> Don't run test
<i>-enable</i> Run test
<i>-quiet</i> Hide the status bar
<i>-sanitize</i> Hide confidential information
<i>-numeric</i> Show numeric IDs

For the sake of completeness, *hdparm* (hardware parameters) should also be mentioned. *hdparm* works with the Linux SATA/PATA/SAS libATA subsystem, the older IDE, and some USB drives released after 2008. It does not work with most solid state drives, and functional options may vary with the kernel. If *hdparm* does work on your system, it allows extensive customization, including setting power management features, 32-bit I/O support on IDE drives, and onboard defect management features. However, the wrong options can crash a system and brick a hard drive, and some options have warnings in the man page. If you can use *hdparm*, it is a powerful tool, but unless you understand exactly what an option does, you are better off avoiding the command altogether.

Using modprobe

modprobe adds or removes modules from the Linux kernel.

```
modprobe [modulename] [option1]=[value1] [option2]=[value2]
# ^^^ basic command structure
modprobe [modulename] --remove
modprobe [modulename] -r
# ^^^ removes a module
modinfo [modulename]
# ^^^ view a list of options for a particular module
modprobe [modulename] -r --dry-run
modprobe [modulename] -rn
# ^^^ see what a command does without actually running it
```

Working with Traditional Configuration Files (source: LINUX SHELL HANDBOOK - 2021 EDITION, page 28)

Although `systemd` is today the norm for launching services at boot time, there are still services traditionally called "daemons," that are shell scripts. Distributions use shell-like configuration files to configure these services easily, without having to invent a new syntax for each configuration. You can usually recognize these configuration files because they have:

- Shell variables (written in capital letters and with no spaces), like `BLUETOOTH_ENABLED = 1` in `/etc/default/bluetooth`
- Comments explaining what each variable means
- Shell functions (sometimes) to extend or override features in existing scripts

In Debian-based systems, most of these config scripts are placed in `/etc/default/*` and can be edited in a text editor as root. If you make changes to any system shell script or write your own system scripts, please keep the following facts in mind:

- If these scripts run as root, they have the potential to destroy your system if you add the wrong command or accidentally activate a command that is supposed to be commented out. To prevent accidents, always backup a configuration file before editing it.
- Scripts are usually called with an "include" to another script by the dot (`.`) shell command, and the calling script will terminate if an `exit` appears.
- In the shell, no spaces are allowed before and after the equal sign (`=`) when setting variables. KDE and Gnome config files frequently have spaces everywhere to make them easier to read, but in the shell, a space means separation of a command and its parameters, which can cause a syntax error. Values or options containing spaces should have quotes around them.

Using systemd (source: LINUX SHELL HANDBOOK - 2021 EDITION, page 29)

`Systemd` began as a replacement for `init` for starting daemons and soon morphed into a general system manager. Configuration files are stored within `/etc/systemd`. General information is stored in nine basic files in `/etc/systemd`, most of which has self-explanatory names, including ones for login, the logfile, and the network. `Systemd` configuration files typically begin with a series of fields in a section labeled `[Manager]` that can be edited freely. The most important of these files is `system.conf`. All the files are detailed in `systemd-user.conf(5)` in the man pages.

For editing individual system resources, `systemd` uses the `systemctl` utility. `systemctl` works with units, or system resources configuration files, usually as a three-part structure:

```
systemctl [sub-command] [unit]
```

For example, the major sub-commands for system services are `status`, `enable`, or `disable`.

Each unit has its own configuration file, rather than being edited like traditional configuration files, unit configuration files can be provided an override file or a full-replacement file by using the `systemctl` command. Overrides are stored in `/etc/systemd/system`. To be used, a new override must be activated with `systemctl daemon-reload`. Running the command `systemd-delta` shows all the overrides currently running on the system. An override marked `[EXTENDED]` shows the location of the override file, whereas one marked `[OVERRIDDEN]` shows the difference between the original and the currently used replacement unit file.

Although controversial when first introduced, systemd is a far more orderly approach than the other methods of configuring or finding information. However it does introduce numerous new concepts that can be overwhelming at first. As with any other methods, be sure you know what you are doing with systemd before you actually edit one of its files. See the article on systemd elsewhere in this issue.

These sources of information and ways to edit are a lot to absorb. Unsurprisingly, functions are often duplicated between different sources. If you change the configuration, usually you should use the same approach consistently, so you can keep track of all the changes made more easily. In many modern systems, the most straightforward approach is to use systemd. However, experienced users often prefer to edit configuration files, a practice that is as old as Linux itself - if not older. Most of the time, what matters is not the method so much as consistency.

Systemd Primer

Systemd has gradually replaced the ancient System V as the leading init system for Linux. Most mainstream Linux variants now use systemd. The init system launches processes when you start your Linux system, and it also stays around to start and stop services while the system is running. Systemd also logs system events, automates processes, and much more.

Anatomy of a Unit File

In systemd parlance, a managed object is known as a *unit*. The files that are used to initialize and start units at boot time are known as *unit files*. Admins will find the unit files in folders such as:

- `/etc/systemd/system/*`
- `/run/systemd/system/*`
- `/usr/lib/systemd/system/*`

Unit files serve a role that is similar to the init scripts of older Linux systems; however, a unit file is not executable. Instead, a unit file is more like a configuration file in the style of Windows *.ini* files.

The `[Unit]` section contains a human-readable description of the service; the `After` variable specifies other services that need to start first. You could use the `Before` variable to declare that the service you are defining with the unit file must start before the service(s) specified with the variable.

The `[Service]` section sets the user account and group that the database server. `Type` determines the boot style. `Simple` means that the program specified below `ExecStart` starts the main process. The two MySQL scripts specified below `ExecStartPre` handle the preparatory work.

`ExecStartPost` calls scripts that need to run after the main program starts. The `mysql-wait-ready` script makes sure MySQL completes the cleanup that it normally performs at start-up time. This means that services that require MySQL do not start until the database is actually ready to accept connections.

Additionally, the unit file sets a timeout and assigns the database service to the `multiuser` target. This target is a special unit that basically assumes the role of the previous runlevel 3 in System V, which starts the system normally in multiuser mode.

More Security

Unit files support a slew of other parameters, including options that provide an easy way for improving the security of your services.

The first of these parameters in the `[Service]` section is:

```
PrivateNetwork=yes
```

This setting completely isolates the service from any networks. The service then only sees a loopback device, and even that does not have a connection to the host's actual loopback device. Of course, this option is not very useful for network-based services.

A word of caution: Sometimes you need a network, even if the need is not apparent at first glance. For instance, a service might perform most of its work locally but use Lightweight Directory Access Protocol (LDAP) to handle authentication. In that case, you need to be sure only users with a user ID below 1000 are authenticated; names need to resolve to UIDs locally through `/etc/passwd` for these accounts.

A second security feature in [Service] is:

```
PrivateTmp=yes
```

If this option is set, the service uses its own `/tmp` directory instead of the global `/tmp`, which protects the service against malicious Symlink and DoS attacks that tend to use `/tmp`. However, keep in mind that some services locate communication sockets in `/tmp` that will not work if they are in a private directory.

The next two options let you prevent services from writing to specific directories or even accessing them in any way:

```
ReadOnlyDirectories=/var  
InaccessibleDirectories=/home
```

Linux provides a means for assigning the privileges traditionally associated with superuser. These privileges are known as *capabilities*, and you can see the list of all available capabilities by viewing the *capabilities* man page:

```
man capabilities
```

Systemd additionally lets you assign specific capabilities to a service or withdraw those capabilities through settings in the unit file. For example, the following line in the [Service] section of the unit file:

```
CapabilityBoundingSet=CAP_CHOWN CAP_KILL
```

defines a whitelist of capabilities that the process must have.

Defining such a whitelist is not always easy. The other option is to take capabilities away from the service. If you prepend the capability with a tilde (~), this capability is explicitly taken away.

You can also use the unit file to limit the resources a service can access. The *setrlimit()* man page lists all restrictable resources. For example, if you set the maximum size of a file (FSIZE) that the service is allowed to generate to 0, as shown in the example below, the service cannot write the file anywhere. If you specify 1 as the maximum number of processes (NPROC) the service is allowed to spawn, the service cannot fork any other processes.

```
LimitNPROC=1  
LimitFSIZE=0
```

You can limit other resources in a similar way.

Monitoring Processes

After your system boots, you might want to know whether all the required services are actually running. The `systemctl` command provides an overview of service status. *Systemctl* lists all booted services with status information. If you only want to see the failed startups, try:

```
systemctl --state=failed
```

For a single service, you can view more detailed information with:

```
systemctl status mysql.service
```

The output shows the exit states of the pre- and post-scripts from the unit file, as well as additional information on the service status.

The status messages can be quite long on a case-by-case basis. Admins can thus use either the *n line number* parameter to limit the number of rows to output or the *o file* parameter to redirect everything to a file.

Starting and Stopping

Sometimes you need to stop or restart individual services after a boot or reboot. *Systemctl* and the *stop*, *start*, *restart*, and *reload* commands can help; for example:

```
systemctl stop mysqld  
systemctl start mysqld
```

A user who wants to start or stop a system service must authenticate. If the process does not respond to the `stop` command, the only way out is:

```
systemctl kill unit_name
```

This command sends a kill signal to each process in the process group, even to those that the parent process forked at a later stage. The effect thus resembles `killall process name`. The `-s` option also lets you send another specific signal to a process, for example, *SIGHUP* to trigger a reload, as shown in the following example:

```
systemctl kill -s HUP --kill-who=main crond.service
```

The `--kill-who` option ensures that only the *main* process receives the signal.

Runlevels

Run Level	Mode	Action
0	Halt	Shuts down system
1	Single-User Mode	Does not configure network interfaces, start daemons, or allow non-root logins
2	Multi-User Mode	Does not configure network interfaces or start daemons.
3	Multi-User Mode with Networking	Starts the system normally.
4	Undefined	Not used/User-definable
5	X11	As runlevel 3 + display manager(X)
6	Reboot	Reboots the system

Commands Related to Runlevels

Command	Description
<code>runlevel</code> or <code>who -r</code>	View the current runlevel.
<code>ls -l /lib/systemd/system/runlevel*</code>	View the mapping of various runlevels to the corresponding systemd targets.
<code>systemctl get-default</code>	View your system's default systemd target.
<code>systemctl list-dependencies graphical.target</code>	View the services that are mapped to your system's default systemd target.
<code>systemctl show -p WantedBy sshd.service</code>	See which systemd target (or runlevel) a service requires to run

Troubleshooting the Linux boot process

Drop into the real TTY terminal

The key combination to drop into the terminal used to be `Ct1` + `Alt` + `F1`. But that was before. It has been changed and fairly recently, with 20.04 I think. Anyway, the key combination is now `Ct1` + `Alt` + `Fn` + `F3` (or `F2`, `F4`, `F5`, or `F6`). To return to the desktop, use `Ct1` + `Alt` + `Fn` + `F1`.

This should work in cases where your interface freezes and nothing else works. It is a real terminal, not a terminal emulator or a terminal multiplexer. This type of terminal is sometimes called a "Glass TTY". The vernacular phrase for doing this is "Dropping down to the terminal" because you are now at a layer below the desktop and window manager. If you are using a terminal emulator on the desktop, if the desktop freezes, so does your terminal emulator, but the real terminal may still be functioning. Every Linux/Unix system has at least one terminal running at all times.

If Ubuntu Locks Up Completely (REISUB)

If it locks up completely, you can REISUB it, which is a safer alternative to just cold rebooting the computer.

REISUB by:

While holding `Alt` + `SysReq (Print Screen)`, type `R` `E` `I` `S` `U` `B`.

```
R: Switch to XLATE mode
E: Send Terminate signal to all processes except for init
I: Send Kill signal to all processes except for init
S: Sync all mounted filesystems
U: Remount filesystems as read-only
B: Reboot
```

Some mneumonics for REISUB:

- A palindrome of the word busier, as in *The system is **busier** than it should be!*
- **R**eboot **E**ven **I**f **S**ystem **U**tterly **B**roken.
- or the classic: **R**aising **E**lephants **I**s **S**o **U**tterly **B**oring

NOTE: There exists a less radical way than rebooting the whole system. If `SysReq` key works, you can kill processes one-by-one using `Alt` + `SysReq` + `F`. The kernel will kill the most "expensive" process each time. If you want to kill all processes for one console, you can issue `Alt` + `SysReq` + `K`.