Introduction to Linux, Unix , and Ubuntu and Installation of Ubuntu

Unix is a multitasking, multiuser computer operating system that exists in many variants. The original Unix was developed at AT&T's Bell Labs research center by Ken Thompson, Dennis Ritchie, and others. From the power user's or programmer's perspective, Unix systems are characterized by a modular design that is sometimes called the "Unix philosophy," meaning the OS provides a set of simple tools that each perform a limited, well-defined function, with a unified filesystem as the main means of communication and a shell scripting and command language to combine the tools to perform complex workflows.

By the most simple definition, **UNIX**is a computer operating system - the base software that controls a computer system and its peripherals. In this sense, UNIX behaves in the same way that the perhaps more familiar PC operating systems **Windows** or **MacOS** behave. It provides the base mechanisms for booting a computer, logging in, running applications, storing and retrieving files, etc.

More specificially, the word "UNIX" refers to a family of operating systems that are related to one or both of the original UNIX operating systems - BSD and SystemV. Examples of modern UNIX operating systems include **IRIX** (from SGI), **Solaris** (from Sun), **Tru64** (from Compaq) and **Linux** (from the Free Software community). Even though these different "flavors" of UNIX have unique characteristics and come from different sources, they all work alike in a number of fundamental ways. If you gain familiarity with any one of these UNIX-based operating systems, you will also have gained at least some familiarity with nearly every other variant of UNIX.

UNIX FUNDAMENTALS

UNIX has been around for a long time (over 30 years). It predates the concept of the personal computer. As such, it was designed from the ground up to be a multi-user, shared, networked operating environment. UNIX has concepts such as **Users**, **Groups**, **Permissions** and **Network-Shared Resources** (such as files, printers, other computer systems, etc.) built-in to the core of its design. This makes UNIX a uniquely powerful and flexible operating system. Along with this power and flexibility comes some unique concepts that make UNIX what it is. These concepts are relatively simple and should be understood to take full advantage of the operating system.

- **Users** - In order to make use of a UNIX system, you must first log in. This requires a **user account**, which consists of:
  - **Username:**
    This is your login name and is how you are identified to the system itself and to other users of the system.
  - **Password:**
    Along with your username, your password grants you access to the system. Don't forget or lose your password. If you write your password down, keep it in a safe place.
  - **Default group:**
    The default group that your username belongs to (see **Groups** below).
  - **Contact info:**
    So that system administrators and other users can contact you if necessary.
  - **Home directory:**
    A directory or "folder" assigned to your username. This grants you access to disk storage. This is where you will keep your files and data.
  - **Default shell:**
    The program which manages your login and command line sessions (covered in detail later)

- **Groups** - A UNIX group is a collection of users - i.e. a list of usernames. Groups provide a mechanism to assign **permissions** (see below) to a list of users all at once. For our purposes, group associations are typically based on which research group or area of study a user is affiliated with. Each user can belong to more than one group.
- **Permissions** - Everything in UNIX is "owned" by both a user and a group. The simplest example of this would be files (but this concept is not limited only to files). By manipulating permissions, the user who owns a file can define which other users and groups can read or modify that file. In this way, users can secure sensitive files from prying eyes and keep themselves or others from accidentially deleting important data.
- **Shared Resources** - UNIX is a networked operating environment at its core. As such, nearly everything that one can access on the local system can also be accessed via the network from remote systems. This includes, among other possibilities, editing and sharing files, running software, or using printers. Even the contents of a UNIX system's display can be manipulated remotely.

  The actions that an individual user is able to perform remotely is defined by the permissions assigned to that user (or any group to which the user belongs) for each of these activities. Some of these things will be discussed in greater detail later on.

UNIX PROCESSES

When a program is started on UNIX, it creates what is known as a "process" on the system. Every process is assigned a unique serial number called its **process id** or **PID** for short. Processes can be created by any user, but can only be destroyed by someone with the permissions to do so - usually the user that created the process or the system administrator. This ensures that the compute jobs you start on the system will not be disturbed by any other user of the system until they complete or you decide to stop them yourself.

Processes and process management becomes important on UNIX systems that are shared between a number of users. The concept of users and PIDs is the main tool by which the available system resources are shared fairly among everybody who needs access to them. Processes can be suspended or given lower priority in cases where one or more users should step out of the way for someone else, but wish to do so without losing their work up to that point.

One further consideration on this topic is the fact that a running UNIX process can spawn "child" processes. For example, any program you run from inside a UNIX shell will be a child process of that shell. Conversely, the shell is the parent process of this child. Child proceses have associated with them both their own process id (PID) as well as their parent's process id (PPID).

Normally this concept of parent and child processes is not something you need to be bothered with as a user. However, it can be useful to understand how UNIX organizes processes if you are trying to keep track of certain system resources (e.g. memory and CPU), if you are working with environment variables, or if you need to track down a rogue program or script. Some of these items will be discussed later so it's good to have a basic idea about what a UNIX process is.

LINUX

Linux is a UNIX-base operating system. Its original creator was a Finnish student name Linus Torvalds, although being ‘open source’ it has change a great deal since its original conception. It belongs to nobody, and is free to download and use. Any changes to it are open for all to adopt, and as a result it has developed into a very powerful OS that is rapidly gaining in popularity worldwide, particularly among those seeking an alternative to Windows.

In 1991, hardware was expanding rapidly, and DOS was the king of operating systems. Software development was slower, and Macs, while better, were also much pricier than PCs. UNIX was growing, but at that time in its history the source code was jealously guarded and expensive to use.
Linus Torvalds was a Helsinki university student who liked playing around with software and computers, and in 1991 he announced the creation of a new core operating system that he had named Linux. It is now one of the most used systems for the PC, and is particularly suitable for businesses with small IT budgets. Linux is free to use and install, and is more reliable than almost all other systems, running for many months and even years without a reboot being necessary.

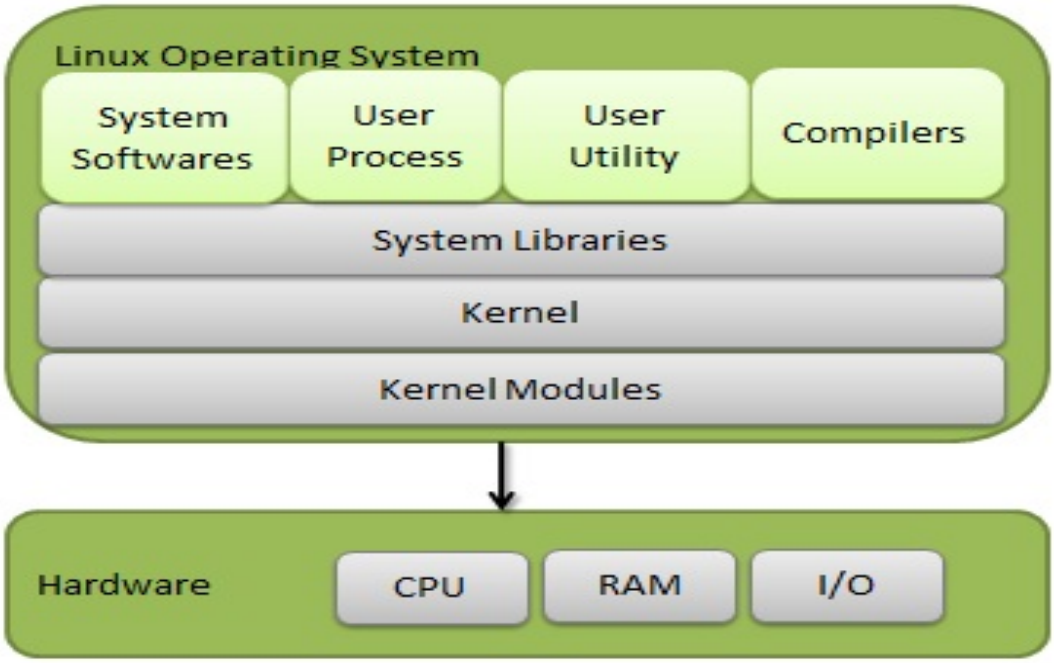
Advantages and Benefits of Linux

One of the significant benefits of open source software such as Linux is that because it has no owner, it can be debugged without resource to a license owner or software proprietor. Businesses therefore have the flexibility to do as they wish with the OS without having to worry about conforming to complex license agreements.
The major advantage of Linux is its cost: the core OS is free, while many software applications also come with a GNU General public License. It can also be used simultaneously by large numbers of users without slowing down or freezing and it is very fast. It is an excellent networking platform and performs at optimum efficiency even with little available hard disk space.
Linux also runs on a wide range of hardware types, including PCs, Macs, mainframes, supercomputers, some cell phones and industrial robots. Some prefer to dual-boot Linux and Windows while others prefer Linux and Mac OS. System76 machines come pre-installed with Linux in the form of Ubuntu, a Debian distribution of Linux. This is the most popular distribution of Linux for laptops.

Components of Linux System

Linux Operating System has primarily three components

- **Kernel** - Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It is consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- **System Library** - System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implements most of the functionalities of the operating system and do not requires kernel module's code access rights.
- **System Utility** - System Utility programs are responsible to do specialized, individual level tasks.



Kernel Mode vs User Mode

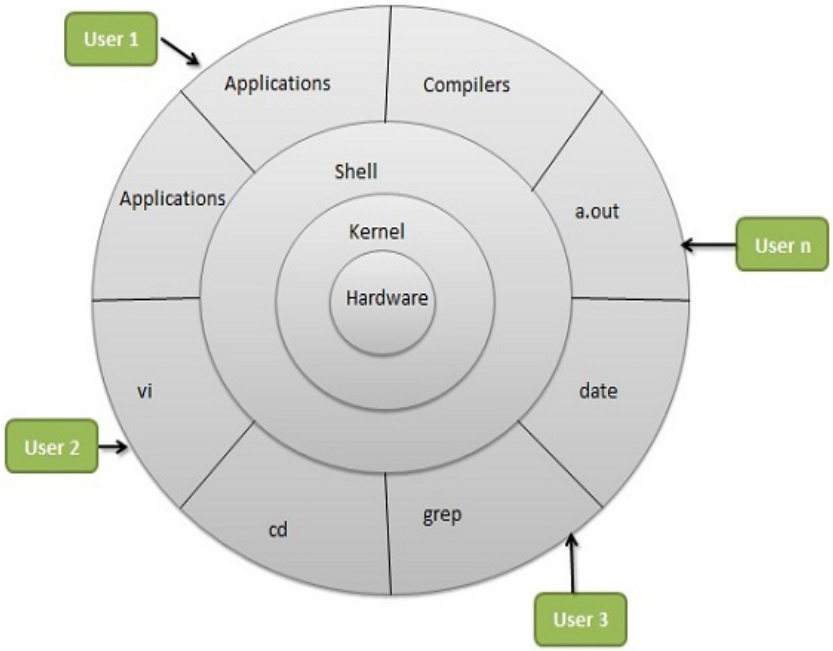
Kernel component code executes in a special privileged mode called **kernel mode** with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each processes and provides system services to processes, provides protected access to hardwares to processes.
Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in **User Mode** which has no access to system hardwares and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks.

Basic Features

Following are some of the important features of Linux Operating System.

- **Portable** - Portability means softwares can works on different types of hardwares in same way.Linux kernel and application programs supports their installation on any kind of hardware platform.
- **Open Source** - Linux source code is freely available and it is community based development project. Multiple teams works in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User** - Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- **Multiprogramming** - Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System** - Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** - Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs etc.
- **Security** - Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Architecture



Linux System Architecture is consists of following layers

- **Hardware layer** - Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
- **Kernel** - Core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
- **Shell** - An interface to kernel, hiding complexity of kernel's functions from users. Takes commands from user and executes kernel's functions.
- **Utilities** - Utility programs giving user most of the functionalities of an operating systems.

Linux Vs Windows

The main benefits and advantages of Linux over other operating systems, particularly Microsoft Windows, are:

- It is free to use and distribute.
- Support is free through online help sites, blogs and forums.
- It is very reliable – more so than most other operating systems with very few crashes.
- A huge amount of free open source software has been developed for it.
- It is very resistant to malware such as spyware, adware and viruses.
- It runs in a wide variety of machines than cannot be updated to use newer Windows versions.
- Since the source code is visible, 'backdoors' are easily spotted, so Linux offers greater security for sensitive applications.
- Linux offers a high degree of flexibility of configuration, and significant customization is possible without modifying the source code.

The Linux operating system is widely use by both home and business users, and its usage is increasing daily. It is considered that Linux will eventually overtake Microsoft Windows as the most popular operating system, which could also open the door further for more free software such as Open Office, The Gimp, Paint, Thunderbird, Firefox and Scribus. It is easy to install and run alongside your existing operating system, so give it a try, because it is also easy to remove if you don't like it – which is unlikely.

UBUNTU

Ubuntu is an operating system that is developed by a worldwide community of programmers as well as by employees of Ubuntu's commercial sponsor, Canonical. Ubuntu is based on the concept of free or open-source software, meaning that you do not pay any licensing fees for Ubuntu, and you can download, use, and share the operating system free of charge.

Being a Linux-based operating system, Ubuntu has a well-deserved reputation for stability and security. Historically, Linux has proven itself to be a workhorse server operating system, and this is where, up until now, it has been most widely used and best known. As of June, 2007, 78 percent of the world's top 500 supercomputers were running Linux, according to Top500.org.

However, in recent years, Linux has also become viable on desktop and laptop computers, making it an option for individuals and businesses. Ubuntu is generally acknowledged to be the most widely used version of Linux available, and Mark Shuttleworth, the founder and CEO of Canonical, estimates Ubuntu has between six and eight million users. Because the software is free to download and share, it is difficult to track exact usage numbers.

Ubuntu versus Windows and OS X

How does Ubuntu compare to the two best-known operating systems — Microsoft Windows and Apple OS X? The most obvious way is in the licensing and distribution terms. Ubuntu is "free software" — a term which is often misunderstood to mean only free of cost. While Ubuntu is free of cost, the term "free software" more accurately refers to the freedom to run the program for any purpose, to study how the program works and modify it to your needs, to redistribute copies, and to improve the program and release your improvements to the public (see the Free Software Foundation's Web site for a detailed definition).

Ubuntu also includes many of the programs used for everyday computing at no cost, unlike Windows and OS X. Some examples are:

- **Office Suite:** OpenOffice.org, a full office suite with a word processor, spreadsheet, and presentation software that can read and write in .doc, .xls, and .ppt formats and can also output to PDF, and supports the ISO standard for electronic office documents, Open Document Format. (Free training for OpenOffice.org is available at LearnFree.org.)
- **Desktop Email Client:** Evolution, an email program with a similar interface to Microsoft Outlook.
- **Web Browser:** Firefox, the increasingly popular Web browser.
- **Databases:** The two best-known open-source databases on Linux are PostgreSQL and MySQL, but commercial databases such as Oracle and IBM's DB2 are also available. There are also tools like Glom that provide an easy-to-use graphical interface for designing and editing databases.
- **Others:** Ubuntu's online Applications Guide lists some Ubuntu-compatible applications that allow you to edit images, listen to and manage music, edit and watch videos, read PDFs, connect to instant messaging services from MSN, AOL, Google, Yahoo, and more.
- **Updates and bug fixes:** Security updates and bug fixes for applications and the operating system are managed by Ubuntu, and users are notified about these updates through an icon in the taskbar, which they can click on to install. (Note that you must be connected to the Internet to receive these notices.)

Another way in which Ubuntu differs from Windows and OS X is in the way it releases new versions. Whereas Apple releases a new version about every 18 months to two years, and Microsoft took nearly

five years between Windows XP and Windows Vista, Ubuntu makes a new version available every six months, which users can update over the Internet without reinstalling the operating system, programs, or settings. (By contrast, neither Windows nor Apple offers online updates, and both require the purchase of a CD/DVD to install.)
Each release includes bug fixes and security updates at no cost for 18 months. After 18 months, security updates and bug fixes will no longer be provided, but you're free to keep using that version of Ubuntu if you like, or update online (free of charge) to a newer version that is supported in this way. Moreover, every two years, Ubuntu releases a version that provides bug fixes and security updates for a longer period of time — three years on desktops or laptops, and five years on servers — making it a good solution for those who want a longer rest between releases.
Of course, being open source gives Ubuntu one other major difference over Windows and OS X, and that is the ability for users to modify it in any way that suits them. There are two types of modifications most relevant here:

- Bug fixes, security fixes, or feature enhancements, which are contributed back to Ubuntu or the original application authors if relevant.
- Customizations to Ubuntu for a given set of circumstances, called a derivatives. Some examples are highlighted below, but a full list can be found on Ubuntu's Web site:
    - nUbuntu, a security-testing platform.
    - Ubuntu Studio, for multi-media editing and creation.

Potential Drawbacks

This all sounds great, but are there any disadvantages to using Ubuntu? Below are a few common challenges Ubuntu users may encounter.

1. Installation

The biggest disadvantage is that whereas Windows and OS X usually come pre-installed on your PC or Mac, a lack of widespread retailers offering pre-installed Ubuntu often means that you must install the system yourself. A technical audience may be familiar with installing operating systems, but the average user will want to avoid this process at all costs.
Ubuntu works hard to make the installation process as easy as possible for the everyday computer user. The installation files for Ubuntu, which you can download and copy to a CD for free, are "live," meaning that you can preview the operating system and applications on your computer directly from the CD without having to install anything onto your hard drive. While your computer may run a little slower because it's running directly from the CD, this offering gives you a good way to test out Ubuntu and determine if it works well with your computers before deciding to install it on your hard drive.
A growing number of retailers are beginning to offer computers pre-installed with Ubuntu, however. The highest-profile vendor to announce this offer in recent months is Dell, but there are other retailers as well, including System76, EmperorLinux, and ZaReason.

2. Hardware Compatibility

Another potential disadvantage related to the lack of widespread pre-installed Ubuntu is the issue of hardware support. A small number of wireless networking cards and display drivers may have issues or reduced functionality because some hardware manufacturers do not release the drivers that are Linux-compatible. This is another area where things are improving as the number of users increases — Intel has very good Linux support, as does their main competitor AMD and many other manufacturers. Again, the Live CD can help determine if a computer's hardware is well supported in Ubuntu before you commit to installing the system.

3. Software Availability

A final area that is a potential problem for people switching to Ubuntu from Windows or Mac OS X is the availability of compatible applications. However, with the exception of commercial games, in almost every case there is an equivalent application available for Linux. Ubuntu comes with a program in its System menu that allows users to browse a directory of available software and install with a single click. Another approach to the issue of application compatibility is a program called Wine, which allows many Windows programs — including iTunes and Photoshop — to run on Linux.

Bottom line? Users should not be put off simply because Ubuntu is different; the differences between Windows XP and Windows Vista, or MS Office 2003 and MS Office 2007, are arguably just as difficult to overcome as any differences between Windows and Ubuntu.

Linux has a reputation for requiring technical wizardry to operate, but Ubuntu has gone to great lengths to make things "just work" as much as possible, and to offer a user interface that's simply laid out and easy to use.

Technical and non-technical workers alike have been able to get started with Ubuntu within the first few minutes of installing the operating system.

The good news is that Ubuntu is no-risk. Ubuntu can be tried out in a number of different ways without removing an existing operating system altogether. In a "dual-boot" configuration, every time the computer starts up, the user is presented with a choice of which operating system to run. A fast computer with a lot of RAM can also run Ubuntu in a virtual machine, which treats the entire operating system as a program.

Is Ubuntu Right for Your Nonprofit?

The philosophy behind Ubuntu, and open source or "Free Software" in general, has similarities with the philanthropic goals of many nonprofit organizations. Both seek to empower people, and in Ubuntu's case they do it by providing people with access to technology that enables them to participate as full members of the digital world at no cost. This attribute may be even more relevant for nonprofits with a technology angle.

If you are providing others with technology, open-source or "Free Software" makes absolute sense, because it defines your right to pass on the technology to others and preserves their right to do the same. It also has relevance if you are capturing and storing data in some way, because it promotes data standards and discourages vendor lock-in. Furthermore, if you are serving a non-English speaking community, Ubuntu might be attractive to you as the desktop environment offers support for 48 languages.

Ubuntu is equally well-suited to the server as it is to the desktop or laptop. It can very capably perform a variety of functions, including:

- **Everyday office tasks**, including Web access, email, instant messaging, and office applications.
- **Web development.** Developers can run a full Web server and development environment on their local workstation to test changes in a similar environment to a production Web server that runs Linux.
- **Server tasks.** File sharing, Web-serving, mail-serving, database-hosting, backups, or any other server task.

How do you determine if Ubuntu is right for your organization? The first thing to consider is what you would like to improve about your current IT setup. If the answer is nothing, then there may be no reason to switch to Ubuntu. However, if you would like to cut licensing costs or are interested in some of the functionality, flexibility, or stability offered by Ubuntu, start by evaluating it in specific areas. There are many factors to bear in mind when considering technology change:

- **How will you manage the migration process?** Before making the switch, do you have the resources to conduct user testing, to evaluate Ubuntu-compatible applications to ensure they provide the features you need, and to determine if it's appropriate for each business function at your organization? (For information on making the switch from Windows to Ubuntu, see the documentation on Ubuntu's Web site.)
- **How will you provide support for the new technology?** In the case of Ubuntu, do you have IT staff or others who are familiar with or comfortable with Linux? Commercial support is also available from Ubuntu's commercial sponsor Canonical for about $250 to $4,000 a year. (See Canonical's Web site for services and pricing.)
- **How will you train users?** Do you have the staff and resources to teach users how to use Ubuntu and the new applications that come with it?
- **Is Ubuntu compatible with your existing programs?** Are there any essential proprietary applications that might make transition to Ubuntu more difficult?

How Are Other Organizations Using Ubuntu?

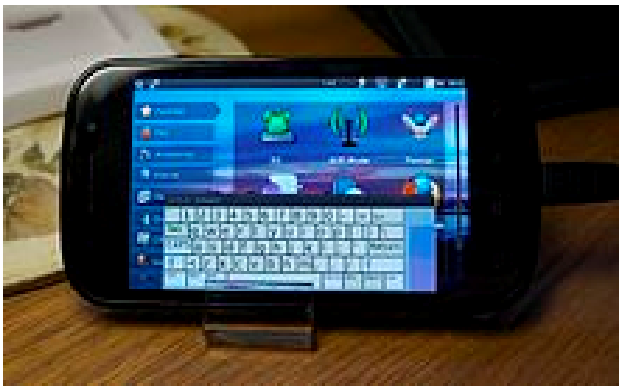
In Andalucía, Spain 185,000 desktop computers in 1,100 schools use Guadalinex, a Spanish-language operating system based on Ubuntu that has been tailored to the local market. Technical support for these 185,000 desktops is provided using approximately 50 overall support and technical staff, including a call center staffed by 16 people.

The John Hopkins University has chosen Ubuntu to be the open-source platform of choice on which to build a support system for technology-enabled classrooms. "We liked that Ubuntu met all our development needs and that we could just download it and get to work," said JHU's Sean Stanley. "We didn't have to engage in any commercial relationship or even register the software. This makes Ubuntu very attractive for developers looking to innovate quickly and cost-effectively. It also combined the right amount of reliability with an unprecedented ease-of-use for Linux. It was a turnkey solution that was inherently secure, something that made the Microsoft users in my group relax."

On a much smaller scale, I've been working with the Hayes Valley Learning Center in San Francisco. A community volunteer approached her local Linux User Group (SF-LUG) and asked for help in setting up a community computer lab using donated computers and Edubuntu, a version of Ubuntu that's tailored for educational environments.

SF-LUG was able to get donated hardware for a server and 15 workstations, install Edubuntu on all of them, and provide a working community lab as well as offer ongoing technical support for the computers in that lab. All workstations are connected to the Internet, and include a full range of educational applications such as typing tutors, math programs, image-editing application GIMP, and constellation mapping programs, among others. Additionally, the lab is set up with content filtering so that inappropriate content can be blocked from younger users, and reports can be reviewed to audit what sites have been blocked when.

Installation



Ubuntu running on the Nexus S, an smartphone that ran Android prior to Ubuntu

The system requirements vary among Ubuntu products. For the Ubuntu desktop release 14.04, a PC with at least 768 MB of RAM and 5 GB of disk space is recommended.[38] For less powerful computers, there are other Ubuntu distributions such as Lubuntu and Xubuntu. As of version 12.04, Ubuntu supports the ARM architecture. Ubuntu is also available on PowerPC, and SPARC platforms, although these platforms are not officially supported.

Live images are the typical way for users to assess and subsequently install Ubuntu. These can be downloaded as a disk image (.iso) and subsequently burnt to a DVD and booted, or run via UNetbootin directly from a USB drive (making, respectively, a live DVD or live USB medium). Running Ubuntu in this way is typically slower than running it from a hard drive, but does not alter the computer unless specifically instructed by the user. If the user chooses to boot the live image rather than execute an installer at boot time, there is still the option to then use an installer called Ubiquity to install Ubuntu once booted into the live environment. Disk images of all current and past versions are available for download at the Ubuntu web site. Various third-party programs such as remastersys and Reconstructor are available to create customized copies of the Ubuntu Live DVDs (or CDs). "Minimal CDs" are available (for server use) that fit on a CD.

Additionally, USB flash drive installations can be used to boot Ubuntu and Kubuntu in a way that allows permanent saving of user settings and portability of the USB-installed system between physical machines (however, the computers' BIOS must support booting from USB). In newer versions of Ubuntu, the Ubuntu Live USB creator can be used to install Ubuntu on a USB drive (with or without a live CD or DVD).

The desktop edition can also be installed using the Netboot image which uses the debian-installer and allows certain specialist installations of Ubuntu: setting up automated deployments, upgrading from older installations without network access, LVM and/or RAID partitioning, installs on systems with less than about 256 MB of RAM (although low-memory systems may not be able to run a full desktop environment reasonably).

The ls Command

ls - list directory contents.

List information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

SYNOPSIS

ls [OPTION]... [FILE]...

DESCRIPTION

Mandatory arguments to long options are mandatory for short options too.

-a, --all
do not ignore entries starting with .

-A, --almost-all
do not list implied . and ..

--author
with -l, print the author of each file

-b, --escape
print C-style escapes for nongraphic characters

--block-size=SIZE
scale sizes by SIZE before printing them. E.g., '--block-size=M' prints sizes in units of 1,048,576 bytes. See SIZE format below.

-B, --ignore-backups
do not list implied entries ending with ~

-c with -lt: sort by, and show, ctime (time of last modification of file status information) with -l: show ctime and sort by name otherwise: sort by
ctime, newest first

-C list entries by columns

-d, --directory
list directory entries instead of contents, and do not dereference symbolic links

-f do not sort, enable -aU, disable -ls --color

-F, --classify
append indicator (one of */=>@|) to entries

--file-type
likewise, except do not append '*'

--format=WORD
across -x, commas -m, horizontal -x, long -l, single-column -1, verbose -l, vertical -C

-g like -l, but do not list owner

--group-directories-first
group directories before files.

augment with a --sort option, but any use of --sort=none (-U) disables grouping

-G, --no-group
in a long listing, don't print group names

-h, --human-readable
with -l, print sizes in human readable format (e.g., 1K 234M 2G)

--hide=PATTERN
do not list implied entries matching shell PATTERN (overridden by -a or -A)

-i, --inode
print the index number of each file

-l use a long listing format

-m fill width with a comma separated list of entries

-o like -l, but do not list group information

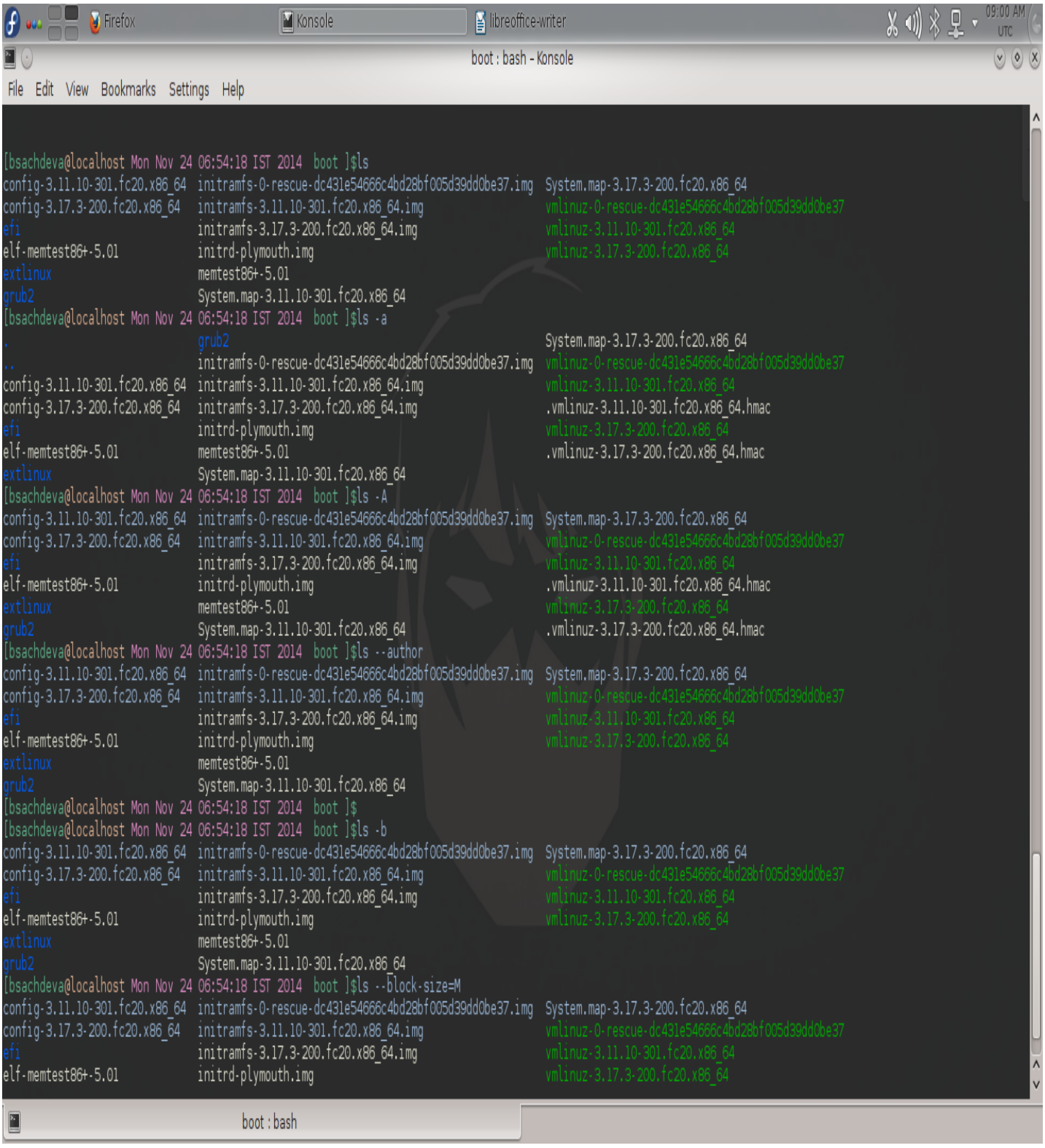
-r, --reverse
reverse order while sorting

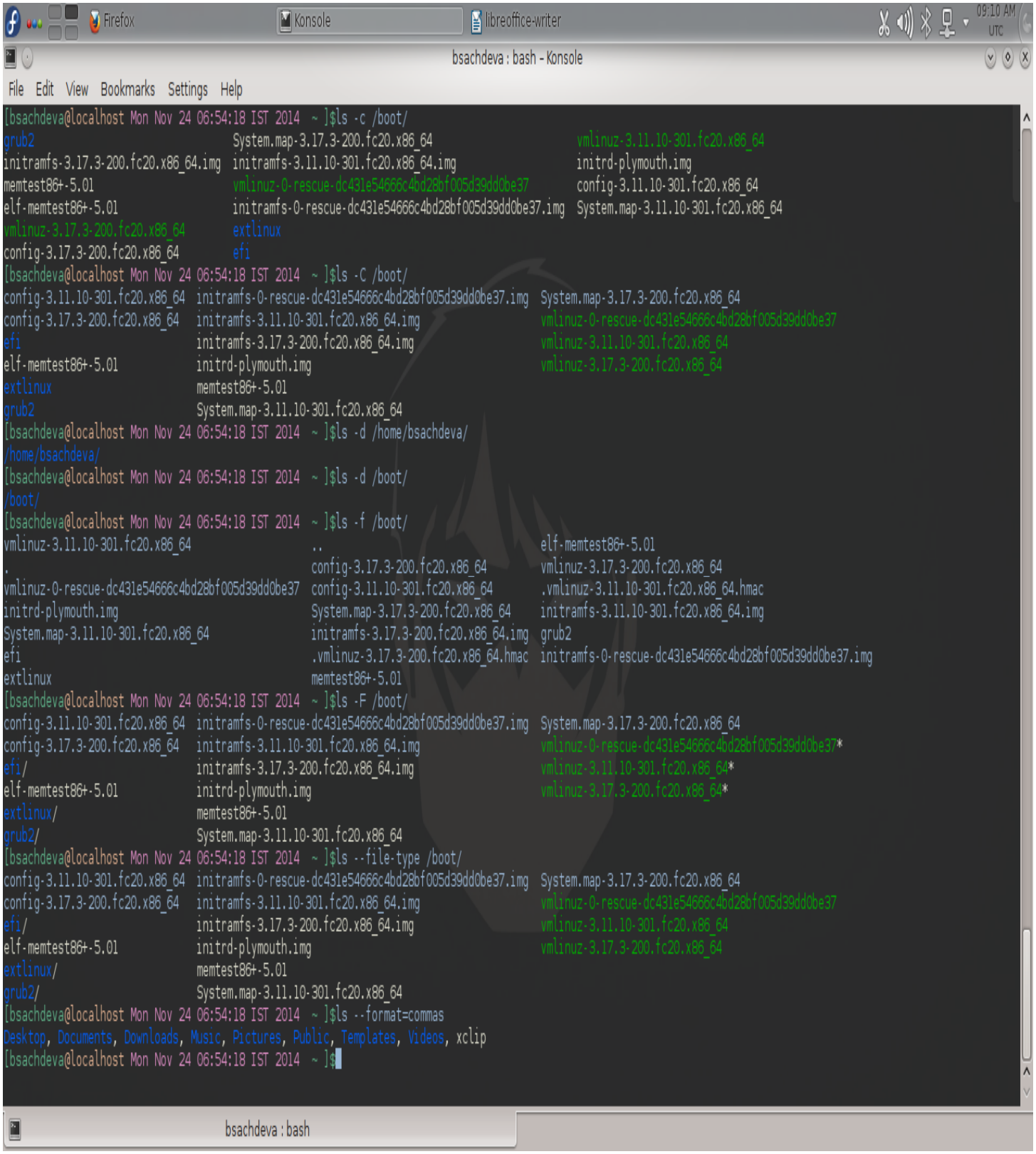
-R, --recursive
list subdirectories recursively

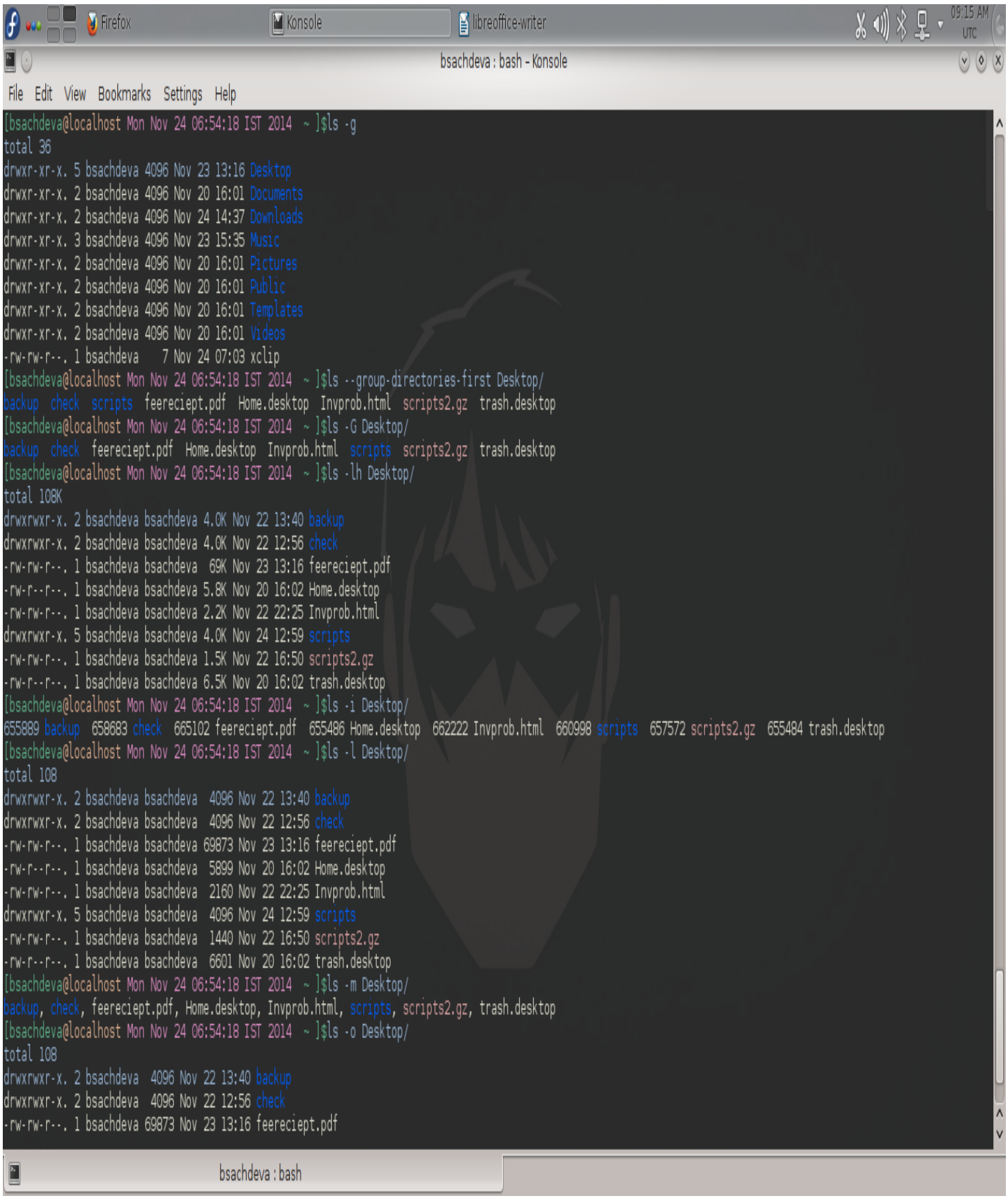
-s, --size
print the allocated size of each file, in blocks

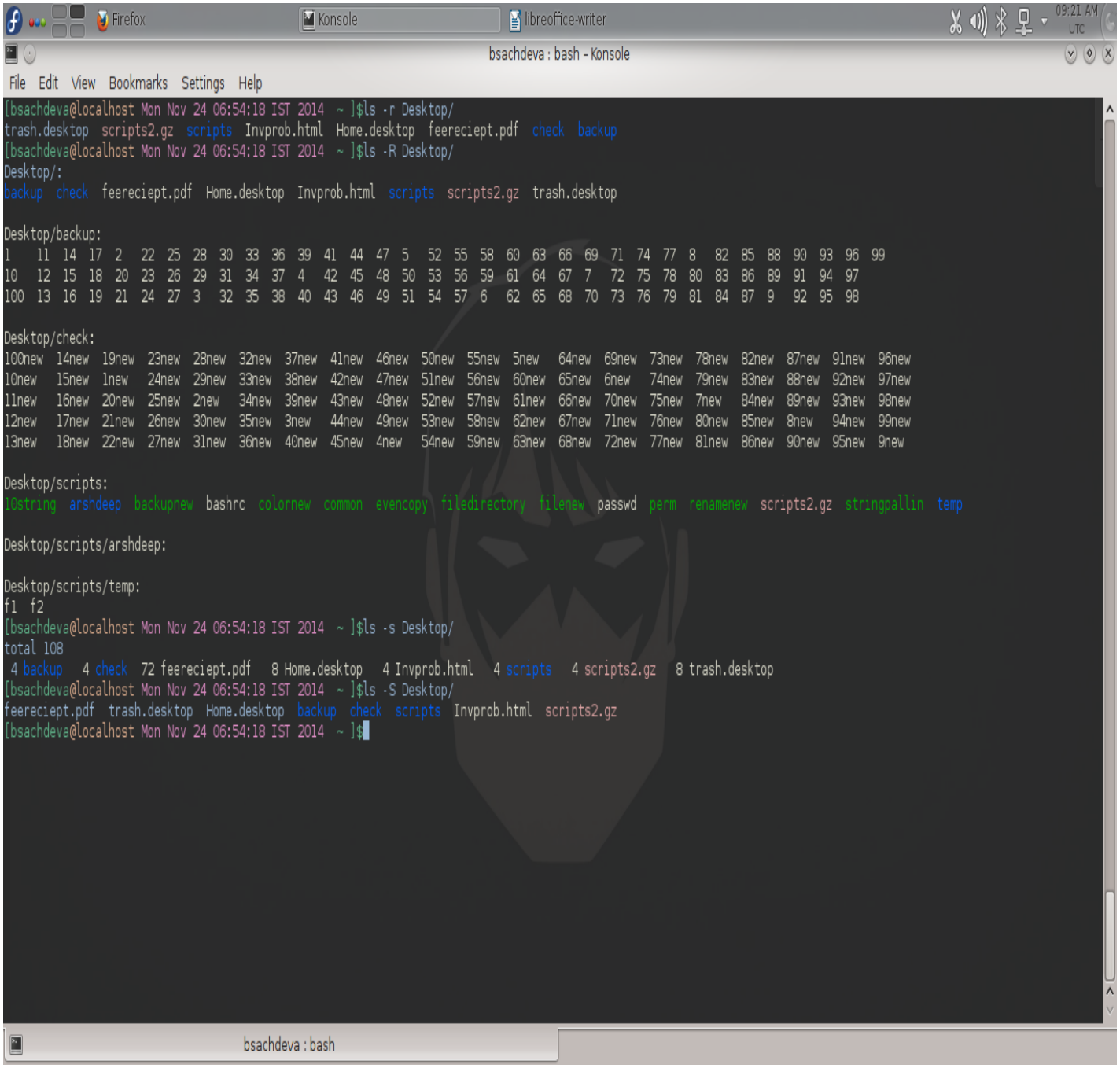
-S sort by file size

Examples


Firefox
Konsole
libreoffice-writer
09:00 AM
UTC
boot : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 boot]$ls
config-3.11.10-301.fc20.x86_64 initramfs-0-rescue-dc431e54666c4bd28bf005d39dd0be37.img System.map-3.17.3-200.fc20.x86_64
config-3.17.3-200.fc20.x86_64 initramfs-3.11.10-301.fc20.x86_64.img vmlinuz-0-rescue-dc431e54666c4bd28bf005d39dd0be37
efi initramfs-3.17.3-200.fc20.x86_64.img vmlinuz-3.11.10-301.fc20.x86_64
elf-memtest86+-5.01 initrd-plymouth.img vmlinuz-3.17.3-200.fc20.x86_64
extlinux memtest86+-5.01
grub2 System.map-3.11.10-301.fc20.x86_64
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 boot]$ls -a
. grub2 System.map-3.17.3-200.fc20.x86_64
.. initramfs-0-rescue-dc431e54666c4bd28bf005d39dd0be37.img vmlinuz-0-rescue-dc431e54666c4bd28bf005d39dd0be37
config-3.11.10-301.fc20.x86_64 initramfs-3.11.10-301.fc20.x86_64.img vmlinuz-3.11.10-301.fc20.x86_64
config-3.17.3-200.fc20.x86_64 initramfs-3.17.3-200.fc20.x86_64.img .vmlinuz-3.11.10-301.fc20.x86_64.hmac
efi initrd-plymouth.img vmlinuz-3.17.3-200.fc20.x86_64
elf-memtest86+-5.01 memtest86+-5.01 .vmlinuz-3.17.3-200.fc20.x86_64.hmac
extlinux System.map-3.11.10-301.fc20.x86_64
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 boot]$ls -A
config-3.11.10-301.fc20.x86_64 initramfs-0-rescue-dc431e54666c4bd28bf005d39dd0be37.img System.map-3.17.3-200.fc20.x86_64
config-3.17.3-200.fc20.x86_64 initramfs-3.11.10-301.fc20.x86_64.img vmlinuz-0-rescue-dc431e54666c4bd28bf005d39dd0be37
efi initramfs-3.17.3-200.fc20.x86_64.img vmlinuz-3.11.10-301.fc20.x86_64
elf-memtest86+-5.01 initrd-plymouth.img .vmlinuz-3.11.10-301.fc20.x86_64.hmac
extlinux memtest86+-5.01 vmlinuz-3.17.3-200.fc20.x86_64
grub2 System.map-3.11.10-301.fc20.x86_64 .vmlinuz-3.17.3-200.fc20.x86_64.hmac
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 boot]$ls --author
config-3.11.10-301.fc20.x86_64 initramfs-0-rescue-dc431e54666c4bd28bf005d39dd0be37.img System.map-3.17.3-200.fc20.x86_64
config-3.17.3-200.fc20.x86_64 initramfs-3.11.10-301.fc20.x86_64.img vmlinuz-0-rescue-dc431e54666c4bd28bf005d39dd0be37
efi initramfs-3.17.3-200.fc20.x86_64.img vmlinuz-3.11.10-301.fc20.x86_64
elf-memtest86+-5.01 initrd-plymouth.img vmlinuz-3.17.3-200.fc20.x86_64
extlinux memtest86+-5.01
grub2 System.map-3.11.10-301.fc20.x86_64
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 boot]$
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 boot]$ls -b
config-3.11.10-301.fc20.x86_64 initramfs-0-rescue-dc431e54666c4bd28bf005d39dd0be37.img System.map-3.17.3-200.fc20.x86_64
config-3.17.3-200.fc20.x86_64 initramfs-3.11.10-301.fc20.x86_64.img vmlinuz-0-rescue-dc431e54666c4bd28bf005d39dd0be37
efi initramfs-3.17.3-200.fc20.x86_64.img vmlinuz-3.11.10-301.fc20.x86_64
elf-memtest86+-5.01 initrd-plymouth.img vmlinuz-3.17.3-200.fc20.x86_64
extlinux memtest86+-5.01
grub2 System.map-3.11.10-301.fc20.x86_64
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 boot]$ls --block-size=M
config-3.11.10-301.fc20.x86_64 initramfs-0-rescue-dc431e54666c4bd28bf005d39dd0be37.img System.map-3.17.3-200.fc20.x86_64
config-3.17.3-200.fc20.x86_64 initramfs-3.11.10-301.fc20.x86_64.img vmlinuz-0-rescue-dc431e54666c4bd28bf005d39dd0be37
efi initramfs-3.17.3-200.fc20.x86_64.img vmlinuz-3.11.10-301.fc20.x86_64
elf-memtest86+-5.01 initrd-plymouth.img vmlinuz-3.17.3-200.fc20.x86_64
boot : bash


Firefox
Konsole
libreoffice-writer
09:10 AM
UTC
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -c /boot/
grub2 System.map-3.17.3-200.fc20.x86_64 vmlinuz-3.11.10-301.fc20.x86_64
initramfs-3.17.3-200.fc20.x86_64.img initramfs-3.11.10-301.fc20.x86_64.img initrd-plymouth.img
memtest86+-5.01 vmlinuz-0-rescue-dc431e54666c4bd28bf005d39dd0be37 config-3.11.10-301.fc20.x86_64
elf-memtest86+-5.01 initramfs-0-rescue-dc431e54666c4bd28bf005d39dd0be37.img System.map-3.11.10-301.fc20.x86_64
vmlinuz-3.17.3-200.fc20.x86_64 extlinux
config-3.17.3-200.fc20.x86_64 efi
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -C /boot/
config-3.11.10-301.fc20.x86_64 initramfs-0-rescue-dc431e54666c4bd28bf005d39dd0be37.img System.map-3.17.3-200.fc20.x86_64
config-3.17.3-200.fc20.x86_64 initramfs-3.11.10-301.fc20.x86_64.img vmlinuz-0-rescue-dc431e54666c4bd28bf005d39dd0be37
efi initramfs-3.17.3-200.fc20.x86_64.img vmlinuz-3.11.10-301.fc20.x86_64
elf-memtest86+-5.01 initrd-plymouth.img vmlinuz-3.17.3-200.fc20.x86_64
extlinux memtest86+-5.01
grub2 System.map-3.11.10-301.fc20.x86_64
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -d /home/bsachdeva/
/home/bsachdeva/
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -d /boot/
/boot/
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -f /boot/
vmlinuz-3.11.10-301.fc20.x86_64 .. elf-memtest86+-5.01
. config-3.17.3-200.fc20.x86_64 vmlinuz-3.17.3-200.fc20.x86_64
vmlinuz-0-rescue-dc431e54666c4bd28bf005d39dd0be37 config-3.11.10-301.fc20.x86_64 .vmlinuz-3.11.10-301.fc20.x86_64.hmac
initrd-plymouth.img System.map-3.17.3-200.fc20.x86_64 initramfs-3.11.10-301.fc20.x86_64.img
System.map-3.11.10-301.fc20.x86_64 initramfs-3.17.3-200.fc20.x86_64.img grub2
efi .vmlinuz-3.17.3-200.fc20.x86_64.hmac initramfs-0-rescue-dc431e54666c4bd28bf005d39dd0be37.img
extlinux memtest86+-5.01
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -F /boot/
config-3.11.10-301.fc20.x86_64 initramfs-0-rescue-dc431e54666c4bd28bf005d39dd0be37.img System.map-3.17.3-200.fc20.x86_64
config-3.17.3-200.fc20.x86_64 initramfs-3.11.10-301.fc20.x86_64.img vmlinuz-0-rescue-dc431e54666c4bd28bf005d39dd0be37*
efi/ initramfs-3.17.3-200.fc20.x86_64.img vmlinuz-3.11.10-301.fc20.x86_64*
elf-memtest86+-5.01 initrd-plymouth.img vmlinuz-3.17.3-200.fc20.x86_64*
extlinux/ memtest86+-5.01
grub2/ System.map-3.11.10-301.fc20.x86_64
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls --file-type /boot/
config-3.11.10-301.fc20.x86_64 initramfs-0-rescue-dc431e54666c4bd28bf005d39dd0be37.img System.map-3.17.3-200.fc20.x86_64
config-3.17.3-200.fc20.x86_64 initramfs-3.11.10-301.fc20.x86_64.img vmlinuz-0-rescue-dc431e54666c4bd28bf005d39dd0be37
efi/ initramfs-3.17.3-200.fc20.x86_64.img vmlinuz-3.11.10-301.fc20.x86_64
elf-memtest86+-5.01 initrd-plymouth.img vmlinuz-3.17.3-200.fc20.x86_64
extlinux/ memtest86+-5.01
grub2/ System.map-3.11.10-301.fc20.x86_64
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls --format=commas
Desktop, Documents, Downloads, Music, Pictures, Public, Templates, Videos, xclip
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$
bsachdeva : bash


Firefox
Konsole
libreoffice-writer
09:15 AM
UTC
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -g
total 36
drwxr-xr-x. 5 bsachdeva 4096 Nov 23 13:16 Desktop
drwxr-xr-x. 2 bsachdeva 4096 Nov 20 16:01 Documents
drwxr-xr-x. 2 bsachdeva 4096 Nov 24 14:37 Downloads
drwxr-xr-x. 3 bsachdeva 4096 Nov 23 15:35 Music
drwxr-xr-x. 2 bsachdeva 4096 Nov 20 16:01 Pictures
drwxr-xr-x. 2 bsachdeva 4096 Nov 20 16:01 Public
drwxr-xr-x. 2 bsachdeva 4096 Nov 20 16:01 Templates
drwxr-xr-x. 2 bsachdeva 4096 Nov 20 16:01 Videos
-rw-rw-r--. 1 bsachdeva 7 Nov 24 07:03 xclip
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls --group-directories-first Desktop/
backup check scripts feereciept.pdf Home.desktop Invprob.html scripts2.gz trash.desktop
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -G Desktop/
backup check feereciept.pdf Home.desktop Invprob.html scripts scripts2.gz trash.desktop
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -lh Desktop/
total 108K
drwxrwxr-x. 2 bsachdeva bsachdeva 4.0K Nov 22 13:40 backup
drwxrwxr-x. 2 bsachdeva bsachdeva 4.0K Nov 22 12:56 check
-rw-rw-r--. 1 bsachdeva bsachdeva 69K Nov 23 13:16 feereciept.pdf
-rw-r--r--. 1 bsachdeva bsachdeva 5.8K Nov 20 16:02 Home.desktop
-rw-rw-r--. 1 bsachdeva bsachdeva 2.2K Nov 22 22:25 Invprob.html
drwxrwxr-x. 5 bsachdeva bsachdeva 4.0K Nov 24 12:59 scripts
-rw-rw-r--. 1 bsachdeva bsachdeva 1.5K Nov 22 16:50 scripts2.gz
-rw-r--r--. 1 bsachdeva bsachdeva 6.5K Nov 20 16:02 trash.desktop
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -i Desktop/
655889 backup 658683 check 665102 feereciept.pdf 655486 Home.desktop 662222 Invprob.html 660998 scripts 657572 scripts2.gz 655484 trash.desktop
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -l Desktop/
total 108
drwxrwxr-x. 2 bsachdeva bsachdeva 4096 Nov 22 13:40 backup
drwxrwxr-x. 2 bsachdeva bsachdeva 4096 Nov 22 12:56 check
-rw-rw-r--. 1 bsachdeva bsachdeva 69873 Nov 23 13:16 feereciept.pdf
-rw-r--r--. 1 bsachdeva bsachdeva 5899 Nov 20 16:02 Home.desktop
-rw-rw-r--. 1 bsachdeva bsachdeva 2160 Nov 22 22:25 Invprob.html
drwxrwxr-x. 5 bsachdeva bsachdeva 4096 Nov 24 12:59 scripts
-rw-rw-r--. 1 bsachdeva bsachdeva 1440 Nov 22 16:50 scripts2.gz
-rw-r--r--. 1 bsachdeva bsachdeva 6601 Nov 20 16:02 trash.desktop
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -m Desktop/
backup, check, feereciept.pdf, Home.desktop, Invprob.html, scripts, scripts2.gz, trash.desktop
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -o Desktop/
total 108
drwxrwxr-x. 2 bsachdeva 4096 Nov 22 13:40 backup
drwxrwxr-x. 2 bsachdeva 4096 Nov 22 12:56 check
-rw-rw-r--. 1 bsachdeva 69873 Nov 23 13:16 feereciept.pdf
bsachdeva : bash


Firefox
Konsole
libreoffice-writer
09:21 AM UTC
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -r Desktop/
trash.desktop scripts2.gz scripts Invprob.html Home.desktop feereciept.pdf check backup
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -R Desktop/
Desktop/:
backup check feereciept.pdf Home.desktop Invprob.html scripts scripts2.gz trash.desktop
Desktop/backup:
1 11 14 17 2 22 25 28 30 33 36 39 41 44 47 5 52 55 58 60 63 66 69 71 74 77 8 82 85 88 90 93 96 99
10 12 15 18 20 23 26 29 31 34 37 4 42 45 48 50 53 56 59 61 64 67 7 72 75 78 80 83 86 89 91 94 97
100 13 16 19 21 24 27 3 32 35 38 40 43 46 49 51 54 57 6 62 65 68 70 73 76 79 81 84 87 9 92 95 98
Desktop/check:
100new 14new 19new 23new 28new 32new 37new 41new 46new 50new 55new 5new 64new 69new 73new 78new 82new 87new 91new 96new
10new 15new 1new 24new 29new 33new 38new 42new 47new 51new 56new 60new 65new 6new 74new 79new 83new 88new 92new 97new
11new 16new 20new 25new 2new 34new 39new 43new 48new 52new 57new 61new 66new 70new 75new 7new 84new 89new 93new 98new
12new 17new 21new 26new 30new 35new 3new 44new 49new 53new 58new 62new 67new 71new 76new 80new 85new 8new 94new 99new
13new 18new 22new 27new 31new 36new 40new 45new 4new 54new 59new 63new 68new 72new 77new 81new 86new 90new 95new 9new
Desktop/scripts:
10string arshdeep backupnew bashrc colornew common evencopy filedirectory filenew passwd perm renamenew scripts2.gz stringpallin temp
Desktop/scripts/arshdeep:
Desktop/scripts/temp:
f1 f2
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -s Desktop/
total 108
4 backup 4 check 72 feereciept.pdf 8 Home.desktop 4 Invprob.html 4 scripts 4 scripts2.gz 8 trash.desktop
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls -S Desktop/
feereciept.pdf trash.desktop Home.desktop backup check scripts Invprob.html scripts2.gz
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$
bsachdeva : bash

The echo command

Echo is a built-in command in the bash and C shells that writes its arguments to standard output. A shell is a program that provides the command line (i.e., the all-text display user interface) on Linux and other Unix-like operating systems. It also executes (i.e., runs) commands that are typed into it and displays the results. bash is the default shell on Linux. A command is an instruction telling a computer to do something. An argument is input data for a command. Standard output is the display screen by default, but it can be redirected to a file, printer, etc.

SYNOPSIS

```
echo [option(s)] [string(s)]
```

The items in square brackets are optional. A string is any finite sequence of characters (i.e., letters, numerals, symbols and punctuation marks).

When used without any options or strings, echo returns a blank line on the display screen followed by the command prompt on the subsequent line. This is because pressing the ENTER key is a signal to the system to start a new line, and thus echo repeats this signal.

When one or more strings are provided as arguments, echo by default repeats those stings on the screen. Thus, for example, typing in the following and pressing the ENTER key would cause echo to repeat the phrase This is a pen. on the screen:

```
echo This is a pen.
```

It is not necessary to surround the strings with quotes, as it does not affect what is written on the screen. If quotes (either single or double) are used, they are not repeated on the screen.

Fortunately, echo can do more than merely repeat verbatim what follows it. That is, it can also show the value of a particular variable if the name of the variable is preceded directly (i.e., with no intervening spaces) by the dollar character ($), which tells the shell to substitute the value of the variable for its name.

For example, a variable named x can be created and its value set to 5 with the following command:

```
x=5
```

The value of x can subsequently be recalled by the following:

```
echo The number is $x.
```

Echo is particularly useful for showing the values of environmental variables, which tell the shell how to behave as a user works at the command line or in scripts (short programs).

For example, to see the value of HOME, the environmental value that shows the current user's home directory, the following would be used:

```
echo $HOME
```

Likewise, echo can be used to show a user's PATH environmental variable, which contains a colon-separated list of the directories that the system searches to find the executable program corresponding to a command issued by the user:

```
echo $PATH
```

echo, by default, follows any output with a newline character. This is a non-printing (i.e., invisible) character that represents the end of one line of text and the start of the next. It is represented by \n in Unix-like operating systems. The result is that the subsequent command prompt begins on a new line rather than on the same line as the output returned by echo.

The -e option is used to enable echo's interpretation of additional instances of the newline character as well as the interpretation of other special characters, such as a horizontal tab, which is represented by \t. Thus, for example, the following would produce a formatted output:

```
echo -e "\n Projects: \n\n\tplan \n\tcode \n\ttest\n"
```

(The above command should be written on a single line, although it may render as two lines on smaller display screens.) The -n option can be used to stop echo from adding the newline to output.

By making use of output redirection, echo provides a very simple way of creating a new file that contains text. This is accomplished by typing echo followed by the desired text, the output redirection operator (which is a rightward pointing angle bracket) and finally the name of the new file. The file can likewise be formatted by using special characters. Thus, for example, the formatted output from the above example could be used to create a new file called project1:

```
echo -e "\n Project1: \n\n\tplan \n\twrite \n\ttest\n" > project1
```

The contents of the new file, including any formatting, can be verified by using a command such as cat or less, i.e.,

```
less project1
```

echo can likewise be a convenient way of appending text to the end of a file by using it together with the the append operator, which is represented by two consecutive rightward pointing angle brackets. However, there is always the risk of accidentally using a single bracket instead of two, thereby overwriting all of the contents of the file, and thus, this feature is best reserved for use in scripts.

echo can also be used with pattern matching, such as the wildcard character, which is represented by the star character. For example, the following would return the phrase The gif files are followed by the names of all the .gif image files in the current directory:

```
echo -e The gif files are *.gif
```

echo is also commonly used to have a shell script display a message or instructions, such as Please enter Y or N in an interactive session with users.

echo is turned off automatically when passwords are entered so that they will not be shown on the screen.

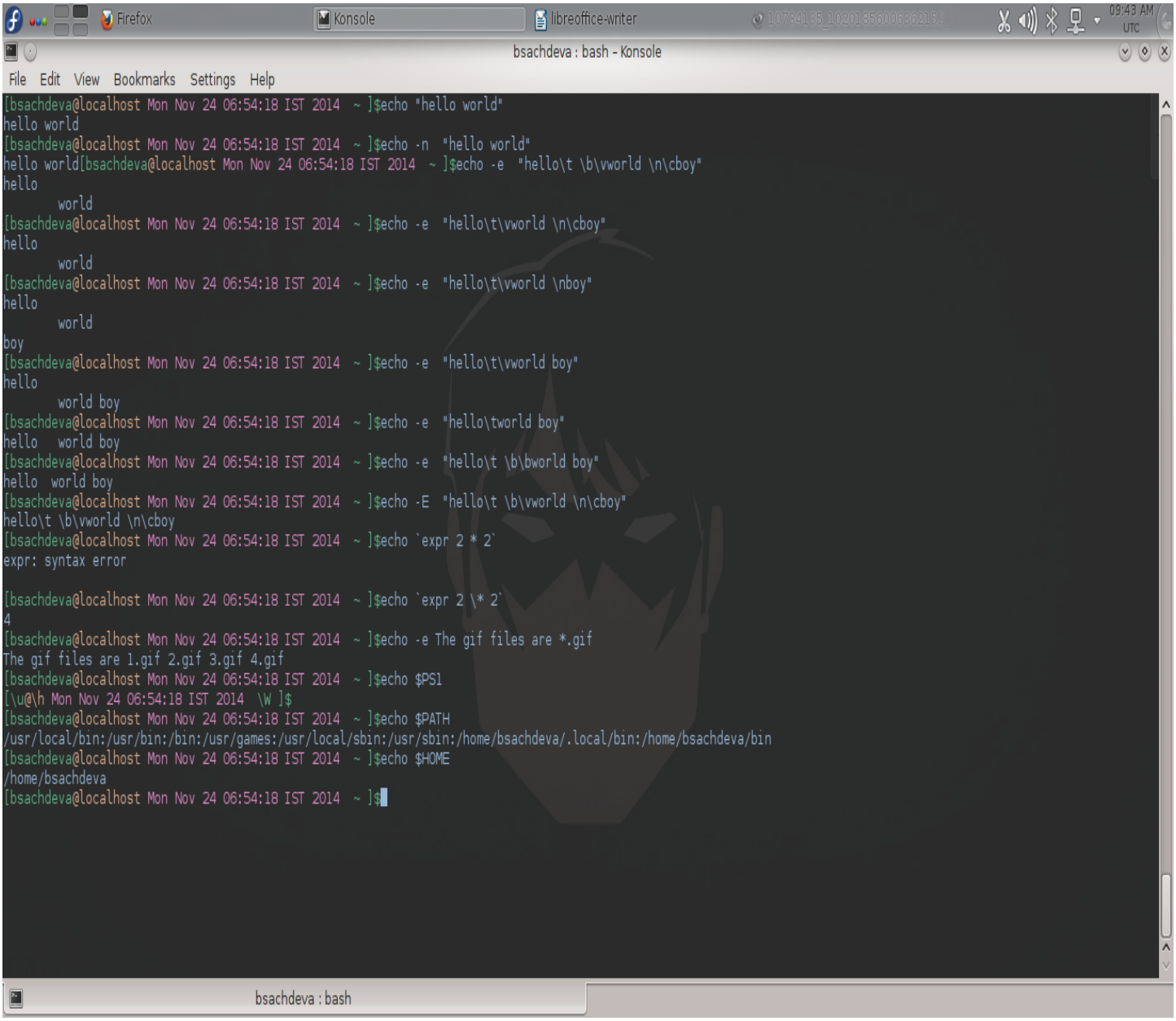
DECRIPTION

Echo the STRING(s) to standard output.

-n do not output the trailing newline

-e enable interpretation of backslash escapes

-E disable interpretation of backslash escapes (default)



The cat Command

cat is one of the most frequently used commands on Unix-like operating systems. It has three related functions with regard to text files: displaying them, combining copies of them and creating new ones.

SYNOPSIS

cat [options] [filenames] [-] [filenames]

The square brackets indicate that the enclosed items are optional.

Reading Files

The most common use of cat is to read the contents of files, and cat is often the most convenient program for this purpose. All that is necessary to open a text file for viewing on the display monitor is to type the word cat followed by a space and the name of the file and then press the ENTER key. For example, the following will display the contents of a file named file1:

```
cat file1
```

The standard output (i.e., default destination of the output) for cat, as is generally the case for other command line (i.e., all-text mode) programs, is the monitor screen. However, it can be redirected from the screen, for example, to another file to be written to that file or to another command to use as the input for that command.

In the following example, the standard output of cat is redirected using the output redirection operator (which is represented by a rightward pointing angular bracket) to file2:

```
cat file1 > file2
```

That is, the output from cat is written to file2 instead of being displayed on the monitor screen.

The standard output could instead be redirected using a pipe (represented by a vertical bar) to a filter (i.e., a program that transforms data in some meaningful way) for further processing. For example, if the file is too large for all of the text to fit on the monitor screen simultaneously, as is frequently the case, the text will scroll down the screen at high speed and be very difficult to read. This problem is easily solved by piping the output to the filter less, i.e.,

```
cat file1 | less
```

This allows the user to advance the contents of the file one screen at a time by pressing the space bar and to move backwards by pressing the b key. The user can exit from less by pressing the q key.

The standard input (i.e., the default source of input data) for cat, as is generally the case for other commands on Unix-like systems, is the keyboard. That is, if no file is specified for it to open, cat will read whatever is typed in on the keyboard.

Typing the command cat followed by the output redirection operator and a file name on the same line, pressing ENTER to move to the next line, then typing some text and finally pressing ENTER again causes the text to be written to that file. Thus, in the following example the text that is typed on the second line will be written to a file named felines:

```
cat > felines
This is not about a feline.
```

The program is terminated and the normal command prompt is restored by pressing the CONTROL and d keys simultaneously.

Repeating the above example without using a redirection operator and specifying a destination file, i.e.,

```
cat
```

```
This is not about a feline.
```

causes the text to be sent to standard output, i.e., to be repeated on the monitor screen.

Concatenation

The second role of cat is concatenation (i.e., stringing together) of copies of the contents of files. (This is the source of cat's curious name.) Because the concatenation occurs only to the copies, there is no effect on the original files.

For example, the following command will concatenate copies of the contents of the three files file1, file2 and file3:

```
cat file1 file2 file3
```

The contents of each file will be displayed on the monitor screen (which, again, is standard output, and thus the destination of the output in the absence of redirection) starting on a new line and in the order that the file names appear in the command. This output could just as easily be redirected using the output redirection operator to another file, such as file4, using the following:

```
cat file1 file2 file3 > file4
```

In the next example, the output of cat is piped to the sort filter in order to alphabetize the lines of text after concatenation and prior to writing to file4:

```
cat file1 file2 file3 | sort > file4
```

File Creation

The third use for cat is file creation. For small files this is often easier than using vi, gedit or other text editors. It is accomplished by typing cat followed by the output redirection operator and the name of the file to be created, then pressing ENTER and finally simultaneously pressing the CONTROL and d keys. For example, a new file named file1 can be created by typing

```
cat > file1
```

then pressing the ENTER key and finally simultaneously pressing the CONTROL and d keys.

If a file named file1 already exists, it will be overwritten (i.e., all of its contents will be erased) by the new, empty file with the same name. Thus the cautious user might prefer to instead use the append operator (represented by two successive rightward pointing angular brackets) in order to prevent unintended erasure. That is,

```
cat >> file1
```

That is, if an attempt is made to create a file by using cat and the append operator, and the new file has the same name as an existing file, the existing file is, in fact, preserved rather than overwritten, and any new text is added to the end of the existing file.

Text can be entered at the time of file creation by typing it in after pressing the ENTER key. Any

amount of text can be typed, including text on multiple lines.

cat can also be used to simultaneously create a new file and transfer to it the data from an existing file. This is accomplished by typing cat, the name of the file from which the output will come, the output redirection operator and the name of the file to be created. Then pressing ENTER causes the new file to be created and written to. For example, typing the following and then pressing ENTER creates a new file named file2 that contains a copy of the contents of file1:

```
cat file1 > file2
```

There is no effect on the contents of file1. (The same thing can, of course, be accomplished just as easily using cp command, which is used to copy files, i.e., cp file1 file2, but the above example does illustrate the great versatility of cat.)

A slight modification to the above procedure makes it possible to create a new file and write text into it from both another file and the keyboard. A hyphen surrounded by spaces is added before the input file if the typed-in text is to come before the text from the input file, and it is added after the input file if the typed-in text is to go after the text from the input file. Thus, for example, to create a new file file6 that consists of text typed in from the keyboard followed by the contents of file5, first enter the following:

```
cat - file5 > file6
```

Or to create a new file file8 that consists of the contents of file7 followed by text typed in from the keyboard, first enter the following:

```
cat file7 - > file8
```

In either case, then press ENTER to move to a new line and type the desired text on any number of lines. Finally press ENTER once more followed by pressing the CONTROL and d keys simultaneously to execute (i.e., run) the command.

An example of a practical application for this use of cat is the creation of form letters (or other documents) for which only the top parts (e.g., dates and names) are customized for each recipient.

DESCRIPTION

-A, --show-all

equivalent to -vET

-b, --number-nonblank

number nonempty output lines, overrides -n

-e equivalent to -vE

-E, --show-ends

display $ at end of each line

-n, --number

number all output lines

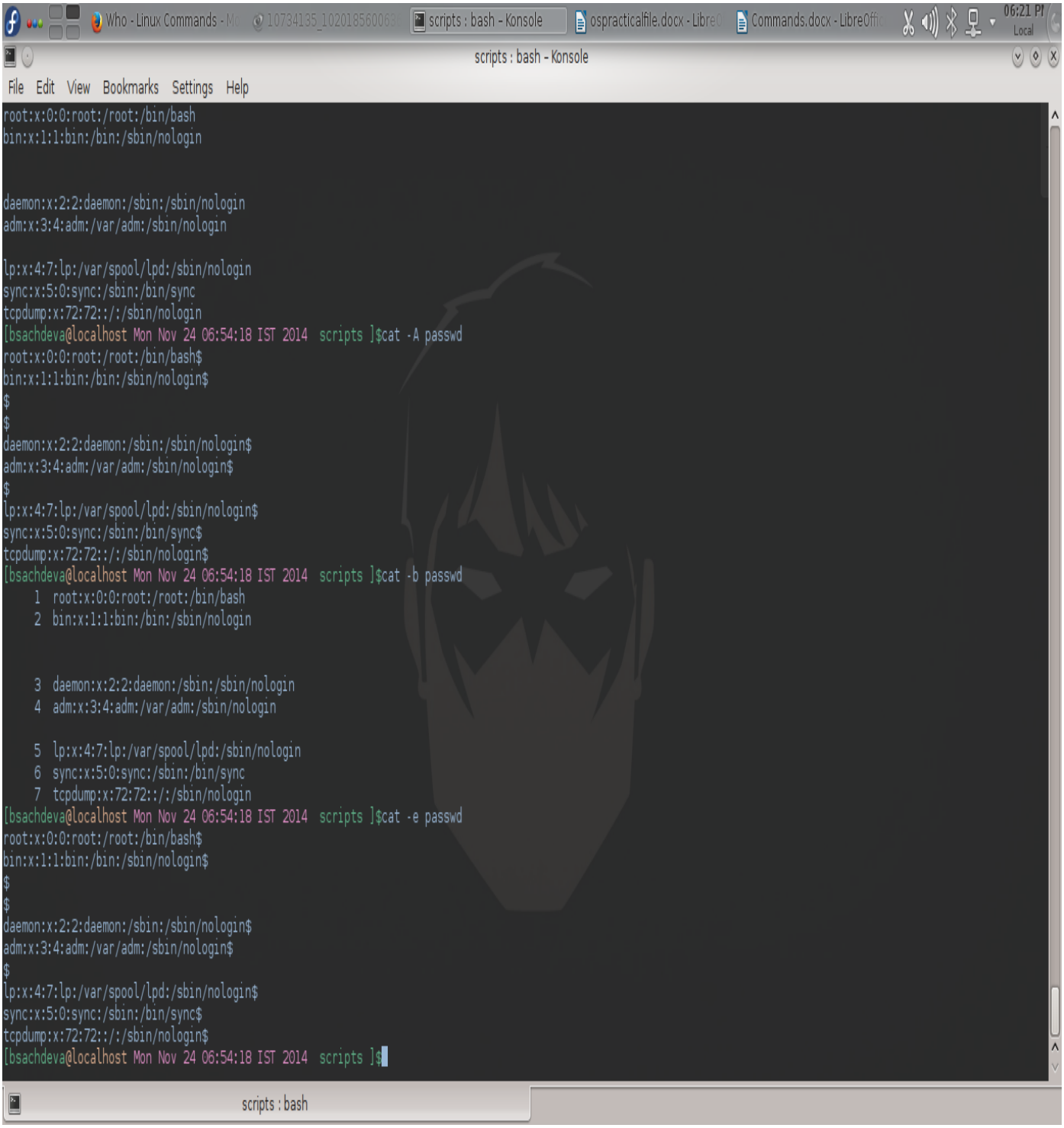
-s, --squeeze-blank

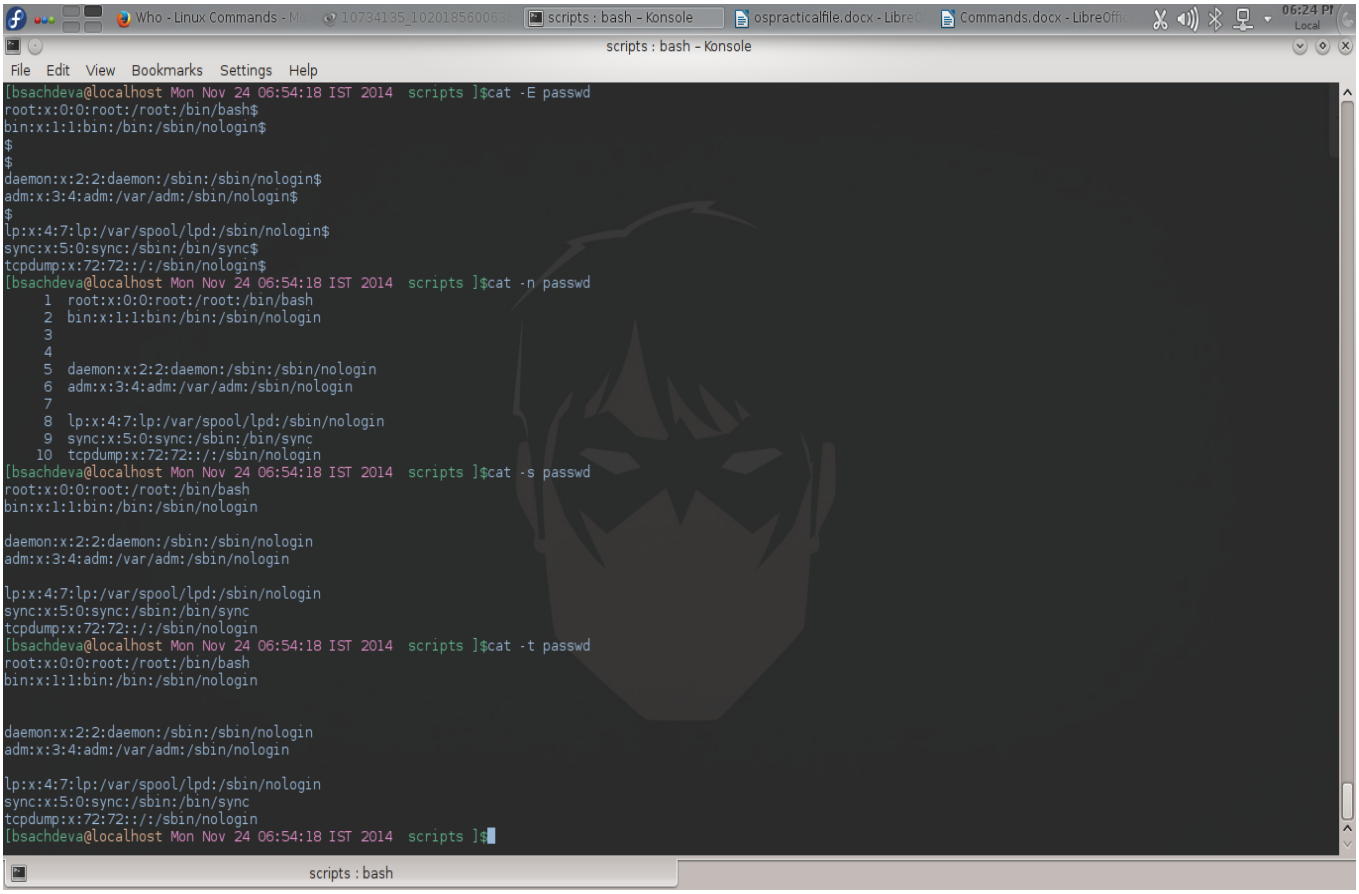
suppress repeated empty output lines

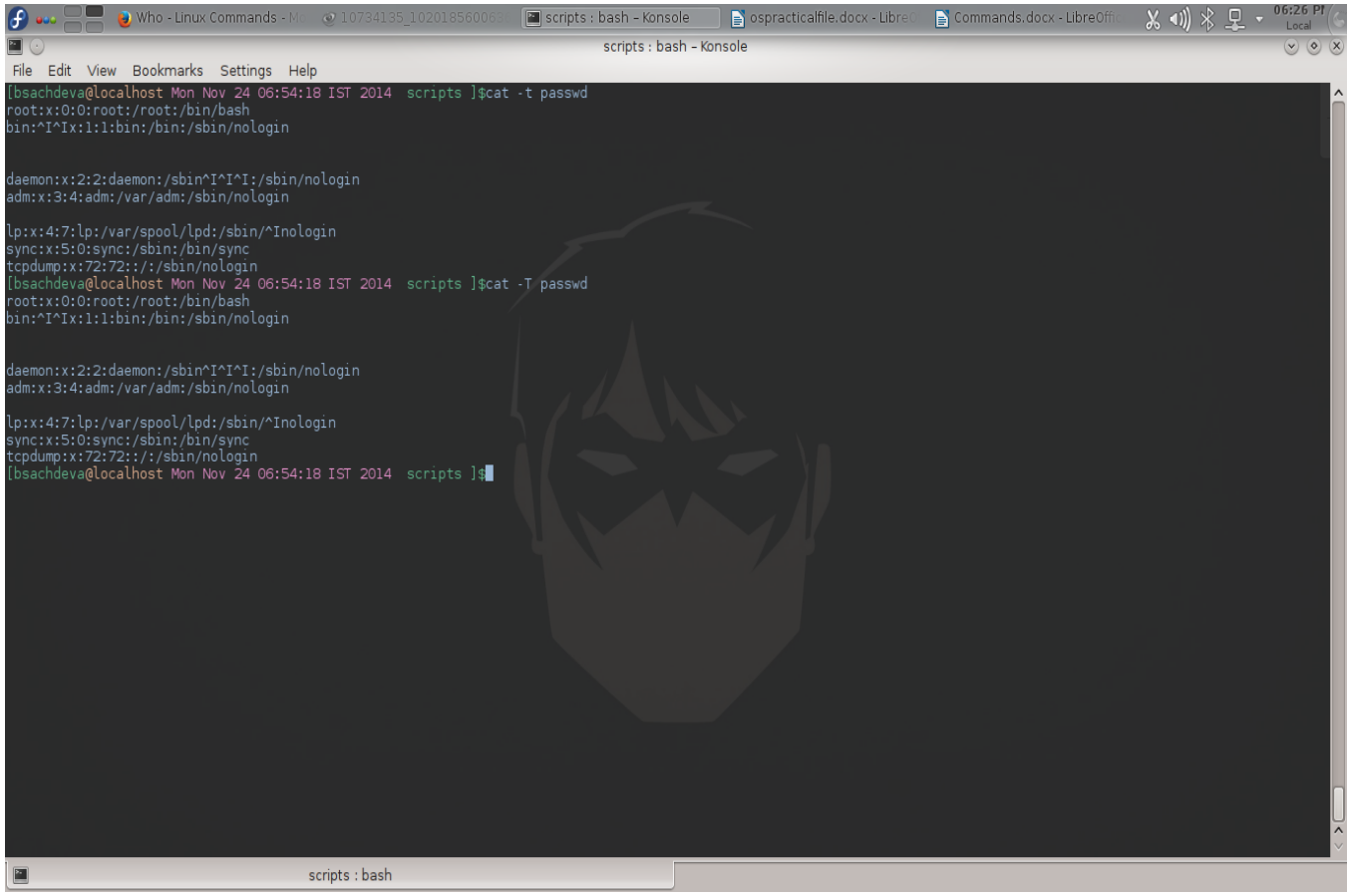
-t equivalent to -vT

-T, --show-tabs

display TAB characters as ^I


scripts : bash - Konsole
File Edit View Bookmarks Settings Help
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
tcpdump:x:72:72::/:/sbin/nologin
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$cat -A passwd
root:x:0:0:root:/root:/bin/bash$
bin:x:1:1:bin:/bin:/sbin/nologin$
$
$
daemon:x:2:2:daemon:/sbin:/sbin/nologin$
adm:x:3:4:adm:/var/adm:/sbin/nologin$
$
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin$
sync:x:5:0:sync:/sbin:/bin/sync$
tcpdump:x:72:72::/:/sbin/nologin$
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$cat -b passwd
1 root:x:0:0:root:/root:/bin/bash
2 bin:x:1:1:bin:/bin:/sbin/nologin
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin
4 adm:x:3:4:adm:/var/adm:/sbin/nologin
5 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
6 sync:x:5:0:sync:/sbin:/bin/sync
7 tcpdump:x:72:72::/:/sbin/nologin
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$cat -e passwd
root:x:0:0:root:/root:/bin/bash$
bin:x:1:1:bin:/bin:/sbin/nologin$
$
$
daemon:x:2:2:daemon:/sbin:/sbin/nologin$
adm:x:3:4:adm:/var/adm:/sbin/nologin$
$
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin$
sync:x:5:0:sync:/sbin:/bin/sync$
tcpdump:x:72:72::/:/sbin/nologin$
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$
scripts : bash


scripts : bash – Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$cat -E passwd
root:x:0:0:root:/root:/bin/bash$
bin:x:1:1:bin:/bin:/sbin/nologin$
$
$
daemon:x:2:2:daemon:/sbin:/sbin/nologin$
adm:x:3:4:adm:/var/adm:/sbin/nologin$
$
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin$
sync:x:5:0:sync:/sbin:/bin/sync$
tcpdump:x:72:72::/:/sbin/nologin$
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$cat -n passwd
1 root:x:0:0:root:/root:/bin/bash
2 bin:x:1:1:bin:/bin:/sbin/nologin
3
4
5 daemon:x:2:2:daemon:/sbin:/sbin/nologin
6 adm:x:3:4:adm:/var/adm:/sbin/nologin
7
8 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
9 sync:x:5:0:sync:/sbin:/bin/sync
10 tcpdump:x:72:72::/:/sbin/nologin
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$cat -s passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
tcpdump:x:72:72::/:/sbin/nologin
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$cat -t passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
tcpdump:x:72:72::/:/sbin/nologin
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$
scripts : bash


scripts : bash – Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$cat -t passwd
root:x:0:0:root:/root:/bin/bash
bin:^I^Ix:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin^I^I^I:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/^Inologin
sync:x:5:0:sync:/sbin:/bin/sync
tcpdump:x:72:72::/:/sbin/nologin
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$cat -T passwd
root:x:0:0:root:/root:/bin/bash
bin:^I^Ix:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin^I^I^I:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/^Inologin
sync:x:5:0:sync:/sbin:/bin/sync
tcpdump:x:72:72::/:/sbin/nologin
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$
scripts : bash

The touch Command

The touch command is the easiest way to create new, empty files. It is also used to change the time stamps (i.e., dates and times of the most recent access and modification) on existing files and directories.

SYNOPSIS

touch [option] file_name(s)

When used without any options, touch creates new files for any file names that are provided as arguments (i.e., input data) if files with such names do not already exist. Touch can create any number of files simultaneously.

Thus, for example, the following command would create three new, empty files named file1, file2 and file3:

touch file1 file2 file3

A nice feature of touch is that, in contrast to some commands such as cp (which is used to copy files and directories) and mv (which is used to move or rename files and directories), it does not automatically overwrite (i.e., erase the contents of) existing files with the same name. Rather, it merely changes the last access times for such files to the current time.

Several of touch's options are specifically designed to allow the user to change the time-stamps for files. For example, the -a option changes only the access time, while the -m option changes only the modification time. The use of both of these options together changes both the access and modification times to the current time, for example:

touch -am file3

The -r (i.e., reference) option followed directly by a space and then by a file name tells touch to use that file's time stamps instead of current time. For example, the following would tell it to use the times of file4 for file5:

touch -r file4 file5

The -B option modifies the timestamps by going back the specified number of seconds, and the -F option modifies the time by going forward the specified number of seconds. For example, the following command would make file7 30 seconds older than file6.

touch -r file6 -B 30 file7

The -d and -t options allow the user to add a specific last access time. The former is followed by a string (i.e., sequence of characters) in the date, month, year, minute:second format, and the latter uses a [[CC]YY]MMDDhhmm[.ss] format. For example, to change the last access time of file8 to 10:22 a.m.

May 1, 2005, 1 May 2005 10:22 would be enclosed in single quotes and used as follows, i.e.,:

```
touch -d '1 May 2005 10:22' file8
```

Partial date-time strings can be used. For example, only the date need be provided, as shown for file9 below (in which case the time is automatically set to 0:00):

```
touch -d '14 May' file9
```

Just providing the time, as shown below, automatically changes the date to the current date:

```
touch -d '14:24' file9
```

The most commonly used way to view the last modification date for files is to use the ls command with its -l option. For example, in the case of a file named file10 this would be

```
ls -l file10
```

The complete timestamps for any file or directory can be viewed by using the stat command. For example, the following would show the timestamps for a file named file11:

```
stat file11
```

The --help option displays a basic list of options, and the --version option returns the version of the currently installed touch program.

DESCRIPTION

Mandatory arguments to long options are mandatory for short options too.

-a change only the access time

-c, --no-create

do not create any files

-d, --date=STRING

parse STRING and use it instead of current time

-h, --no-dereference

affect each symbolic link instead of any referenced file (useful only on systems that can change the timestamps of a symlink)

-m change only the modification time

-r, --reference=FILE

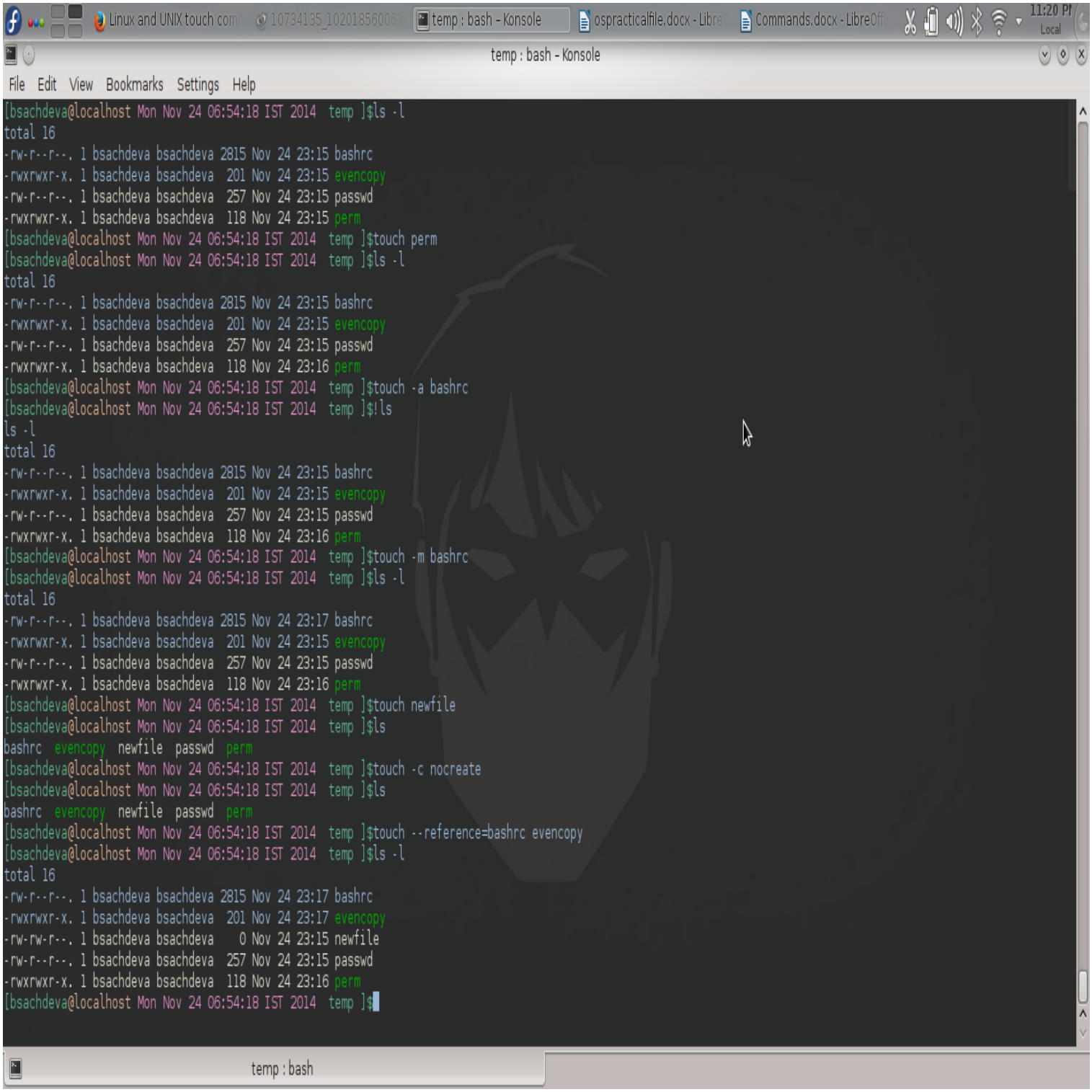
use this file's times instead of current time

-t STAMP

use [[CC]YY]MMDDhhmm[.ss] instead of current time

--time=WORD

change the specified time: WORD is access, atime, or use: equivalent to -a WORD is modify or mtime: equivalent to -m



The man Command

The man command is used to format and display the man pages. It is an interface to the on-line reference manuals.

SYNOPSIS

The man pages are a user manual that is by default built into most Linux distributions (i.e., versions) and most other Unix-like operating systems during installation. They provide extensive documentation about commands and other aspects of the system, including configuration files, system calls, library routines and the kernel (i.e., the core of the operating system). A configuration file is a type of simple database that contains data that tells a program or operating system how to behave. A system call is a request made via a software interrupt (i.e., a signal to the kernel initiated by software) by an active process for a service performed by the kernel. A library routine is a subprogram that is used by programmers to simplify the development of software.

The man pages are tailored to the particular operating system, and version thereof, on which they are installed. This is beneficial because there can be slight differences in commands and other items according to the particular system.

The descriptions are rather terse, and they can seem somewhat cryptic to new users. However, users typically find them to be increasingly useful as they become more familiar with them and gain experience in the use of Unix-like operating systems.

The man command itself is extremely easy to use. Its basic syntax is

```
man [option(s)] keyword(s)
```

man is most commonly used without any options and with only one keyword. The keyword is the exact name of the command or other item for which information is desired. For example, the following provides information about the ls command (which is used to list the contents of any specified directory):

```
man ls
```

As another example, the following displays the man page about the man pages:

```
man man
```

man automatically sends its output through a pager, usually the program less. A pager is a program that causes the output of any program to be displayed one screenful at a time, rather than having a large amount of text scroll down the screen at high (and generally unreadable) speed.

less writes a colon at the bottom of the screen to indicate the end of the on-screen page. The user can

move to the next page by pushing the space bar and can return to the previous page by pressing the b key. Pressing the q exits the man pages and returns the user to the shell program.

Each man page is a self-contained article that is divided into a number of sections, the headers for which are labeled with upper case letters. The sections for commands are typically something like NAME, SYNOPSIS, DESCRIPTION, OPTIONS, AUTHOR, BUGS, COPYRIGHT, HISTORY and SEE ALSO, although there may be some differences according to the particular command. Some of these might be broken down into subsections, particularly OPTIONS in the case of a command that has numerous options.

Also, the man pages as a whole are organized into sections, each containing pages about a specific category of topics as shown below. The section to which an article belongs is indicted in parenthesis in the top line, before the NAME header.

1. executable programs or shell commands
2. system calls
3. library routines
4. special files (i.e., devices in the /dev directory)
5. file formats
6. games
7. macro packages
8. system administration commands
9. kernel routines

n. Tcl/Tk (a programming language)

Some topic names will have multiple articles, depending on context. For instance, there are two articles for mount, one corresponding to its use as a command in system management (i.e., to logically attach partition or other devices to the main filesystem) and the other for use in the C programming language. Generally, the most commonly used topic is displayed by default, and there are references to any other topics with the same name in the SEE ALSO section at the bottom of the final on-screen page.

The syntax to specify an article from a particular section is:

man section_number keyword

Thus, for example, the following would display the article about mount from Section 2 instead of from the default Section 8:

man 2 mount

The -w and -W options tell man to not actually display the man pages, but to provide the location(s) of the file(s) that would be formatted or displayed. If no arguments (i.e., input files) are provided, a list of

directories that is searched by man for man pages is returned.

The -f option produces the same output as the whatis command. whatis provides very brief descriptions of commands from a database that is automatically created from the first line of the NAME section of each relevant man page. The -h option displays a terse summary of man's syntax and options.

A simpler version of a man page, i.e., without backspaces and underscores, can be obtained by piping (i.e., transferring) its output to the col command used with its -b option. Thus, for example, the following would write such a version of the man page about the pstree command (which shows the processes currently on the system in a tree diagram) to a text file called pstree.txt (and create a file with this name if it did not already exist):

```
man pstree | col -b > pstree.txt
```

Unix-like operating systems often also have an additional built-in manual referred to as the Info documents, the content of which is largely identical to that of the man pages. These documents can be accessed with the info command.

Although the man pages are usually viewed in a console (i.e., all-text mode) or terminal window (i.e., a text-mode window in a GUI), they can also be viewed in the Konquerer web browser, which is included with many Linux distributions. Some users might find that this provides enhanced usability, including the ability to scroll up and down an entire article regardless of its length and greater ease of changing font sizes. Any man article can be displayed in Konquerer by typing man followed by a colon and the name of the program.

Thus, for example, the following would tell Konquerer to display the man page for pstree:

```
man:pstree
```

The Linux Information Project (LINFO) is providing an on-line alternative to the man pages with its series of articles about commonly used commands and other aspects of Linux. One major difference is that the LINFO articles are designed to be useful to users of all levels, including absolute beginners. In particular, emphasis is placed on the most useful options for commands and specific examples are often provided. An index, along with brief descriptions, of such articles about commands that have been provided to date is Index of Linux Commands.


Linux and UNIX touch comm
10734135_10201856006
man : man - Konsole
ospracticalfile.docx - Libre
Commands.docx - LibreOffi
11:25 PM
Local
man : man - Konsole
File Edit View Bookmarks Settings Help
MAN(1) Manual pager utils MAN(1)
NAME
man - an interface to the on-line reference manuals
SYNOPSIS
man [-C file] [-d] [-D] [--warnings[=warnings]] [-R encoding] [-L locale] [-m system[,...]] [-M path] [-S list] [-e extension] [-i|-I] [--regex|--wildcard]
[--names-only] [-a] [-u] [--no-subpages] [-P pager] [-r prompt] [-7] [-E encoding] [--no-hyphenation] [--no-justification] [-p string] [-t] [-T[device]]
[-H[browser]] [-X[dpi]] [-Z] [[section] page ...] ...
man -k [apropos options] regexp ...
man -K [-w|-W] [-S list] [-i|-I] [--regex] [section] term ...
man -f [whatis options] page ...
man -l [-C file] [-d] [-D] [--warnings[=warnings]] [-R encoding] [-L locale] [-P pager] [-r prompt] [-7] [-E encoding] [-p string] [-t] [-T[device]]
[-H[browser]] [-X[dpi]] [-Z] file ...
man -w|-W [-C file] [-d] [-D] page ...
man -c [-C file] [-d] [-D] page ...
man [-?V]
DESCRIPTION
man is the system's manual pager. Each page argument given to man is normally the name of a program, utility or function. The manual page associated with
each of these arguments is then found and displayed. A section, if provided, will direct man to look only in that section of the manual. The default action
is to search in all of the available sections following a pre-defined order ("1 1p 8 2 3 3p 4 5 6 7 9 0p n l p o 1x 2x 3x 4x 5x 6x 7x 8x" by default, unless
overridden by the SECTION directive in /etc/man_db.conf), and to show only the first page found, even if page exists in several sections.
The table below shows the section numbers of the manual followed by the types of pages they contain.
1 Executable programs or shell commands
2 System calls (functions provided by the kernel)
3 Library calls (functions within program libraries)
4 Special files (usually found in /dev)
5 File formats and conventions eg /etc/passwd
6 Games
7 Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
8 System administration commands (usually only for root)
9 Kernel routines [Non standard]
A manual page consists of several sections.
Conventional section names include NAME, SYNOPSIS, CONFIGURATION, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUE, ERRORS, ENVIRONMENT, FILES, VERSIONS,
CONFORMING TO, NOTES, BUGS, EXAMPLE, AUTHORS, and SEE ALSO.
The following conventions apply to the SYNOPSIS section and can be used as a guide in other sections.
Manual page man(1) line 1 (press h for help or q to quit)
man : man

The mkdir Command

The mkdir command is is used to create new directories.A directory, referred to as a folder in some operating systems, appears to the user as a container for other directories and files. However, Unix-like operating systems treat directories as merely a special type of file that contains a list of file names and their corresponding inode numbers. Each inode number refers to an inode, which is located in inode tables (which are kept at strategic locations around the filesystem) and which contains all information about a file (e.g., size, permissions and date of creation) except its name and the actual data that the file contains.

SYNOPSIS

mkdir [option] directory_name(s)

directory_name is the name of any directory that the user is asking mkdir to create. Any number of directories can be created simultaneously.

Thus, for example, the following command would create three directories within the current directory (i.e., the directory in which the user is currently working) with the names dir_1, dir_2 and dir_3:

mkdir dir_1 dir_2 dir_3

If a directory name provided as an argument (i.e., input) to mkdir is the same as that of an existing directory or file in the same directory in which the user is asking mkdir to create the new directory, mkdir will return a warning message such as mkdir: cannot create directory `dir_1': File exists and will not create a file with that name. However, it will then continue to create directories for any other names provided as arguments.

It is necessary for a user to have write permission (i.e., permission from the system to create or change a file or directory) in the parent directory (i.e., the directory in which the new directory is to be created) in order to be able to create a new directory.

Directories created by mkdir automatically include two hidden directories, one representing the directory just created (and represented by a single dot) and the other representing its parent directory (and represented by two consecutive dots). This can be seen by using the ls (i.e., list) command with its -a option, which tells ls to show all directories and files, (including hidden ones) in any directory provided to it as an argument, or in the current directory if there are no arguments, i.e.,

ls -a

mkdir's -m option is used to control the permissions of new directories. New directories are by default created with the read, write and execute (i.e., run as a program if a program) permissions enabled for the owner (i.e., the creator of the directory by default) and group and the read and execute permissions enabled for other users. Thus, for example, to create a directory named dir_4 for which all three types of permissions were enabled for all users, the sequence 777 would be employed after -m, for example:

```
mkdir -m 777 dir_4
```

The first digit represents the owner, the second represents the group and the third represents other users. The number 7 represents all three types of permission (i.e., read, write and execute), 6 stands for read and write only, 5 stands for read and execute, 4 is read only, 3 is write and execute, 2 is write only, 1 is execute only and 0 is no permissions.

Thus, for example, to create a new directory named dir_5 for which the owner has read and write permissions, the group has read permission and other users have no permissions, the following would be used:

```
mkdir -m 640 dir_5
```

The -p (i.e., parents) option creates the specified intermediate directories for a new directory if they do not already exist. For example, it can be used to create the following directory structure:

```
mkdir -p food/fruit/citrus/oranges
```

It is very easy to confirm that this series of directories has been created by using the du (i.e., disk usage) command with the name of the first directory as an argument. In the case of the above example this would be

```
du food
```

Other options include -v (i.e., verbose), which returns a message for each created directory, --help, which returns brief information about mkdir, and --version, which returns the version number of the currently installed mkdir program.

Additional information about mkdir can be obtained from the mkdir man page.

Directories can be removed with the rmdir and rm commands.

DESCRIPTION

-m, --mode=MODE

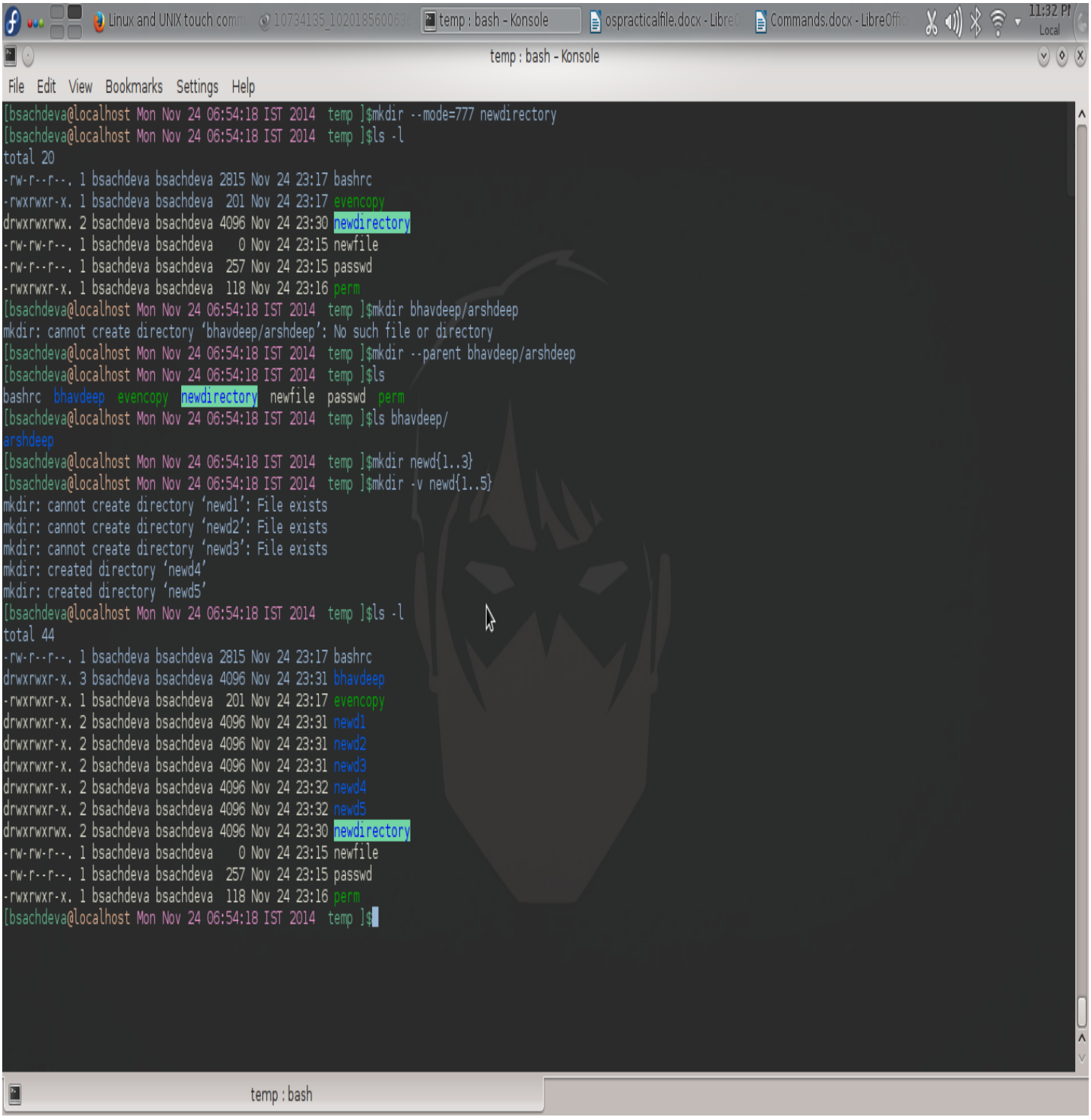
set file mode (as in chmod), not a=rwx - umask

-p, --parents

no error if existing, make parent directories as needed

-v, --verbose

print a message for each created directory


Linux and UNIX touch comm
temp : bash - Konsole
ospracticalfile.docx - Libre
Commands.docx - LibreOffic
11:32 PI
Local
temp : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 temp]$mkdir --mode=777 newdirectory
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 temp]$ls -l
total 20
-rw-r--r--. 1 bsachdeva bsachdeva 2815 Nov 24 23:17 bashrc
-rwxrwxr-x. 1 bsachdeva bsachdeva 201 Nov 24 23:17 evencopy
drwxrwxrwx. 2 bsachdeva bsachdeva 4096 Nov 24 23:30 newdirectory
-rw-rw-r--. 1 bsachdeva bsachdeva 0 Nov 24 23:15 newfile
-rw-r--r--. 1 bsachdeva bsachdeva 257 Nov 24 23:15 passwd
-rwxrwxr-x. 1 bsachdeva bsachdeva 118 Nov 24 23:16 perm
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 temp]$mkdir bhavdeep/arshdeep
mkdir: cannot create directory ‘bhavdeep/arshdeep’: No such file or directory
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 temp]$mkdir --parent bhavdeep/arshdeep
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 temp]$ls
bashrc bhavdeep evencopy newdirectory newfile passwd perm
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 temp]$ls bhavdeep/
arshdeep
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 temp]$mkdir newd{1..3}
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 temp]$mkdir -v newd{1..5}
mkdir: cannot create directory ‘newd1’: File exists
mkdir: cannot create directory ‘newd2’: File exists
mkdir: cannot create directory ‘newd3’: File exists
mkdir: created directory ‘newd4’
mkdir: created directory ‘newd5’
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 temp]$ls -l
total 44
-rw-r--r--. 1 bsachdeva bsachdeva 2815 Nov 24 23:17 bashrc
drwxrwxr-x. 3 bsachdeva bsachdeva 4096 Nov 24 23:31 bhavdeep
-rwxrwxr-x. 1 bsachdeva bsachdeva 201 Nov 24 23:17 evencopy
drwxrwxr-x. 2 bsachdeva bsachdeva 4096 Nov 24 23:31 newd1
drwxrwxr-x. 2 bsachdeva bsachdeva 4096 Nov 24 23:31 newd2
drwxrwxr-x. 2 bsachdeva bsachdeva 4096 Nov 24 23:31 newd3
drwxrwxr-x. 2 bsachdeva bsachdeva 4096 Nov 24 23:32 newd4
drwxrwxr-x. 2 bsachdeva bsachdeva 4096 Nov 24 23:32 newd5
drwxrwxrwx. 2 bsachdeva bsachdeva 4096 Nov 24 23:30 newdirectory
-rw-rw-r--. 1 bsachdeva bsachdeva 0 Nov 24 23:15 newfile
-rw-r--r--. 1 bsachdeva bsachdeva 257 Nov 24 23:15 passwd
-rwxrwxr-x. 1 bsachdeva bsachdeva 118 Nov 24 23:16 perm
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 temp]$
temp : bash

The cd Command

The cd command is used to change the current directory (i.e., the directory in which the user is currently working) in Linux and other Unix-like operating systems. It is similar to the CD and CHDIR commands in MS-DOS.

SYNOPSIS

cd [option] [directory]

The items in square brackets are optional. When used without specifying any directory name, cd returns the user to the previous current directory. This provides a convenient means of toggling between two directories.

When a directory name is provided, cd changes the current directory to it. The name can be expressed as an absolute pathname (i.e., location relative to the root directory) or as a local pathname (i.e., location relative to the current directory). It is usually more convenient to use a local pathname when changing to a subdirectory of the current directory.

As an example, the following would change the current directory, regardless of where it is on the system (because it is an absolute path), to the root directory (which is represented by a forward slash):

cd /

Likewise, the following would change the current directory, regardless of its location, to the /usr/sbin directory (which contains non-vital system utilities that are used by the system administrator):

cd /usr/sbin

If a user currently in the directory /usr/local/share/man/ desired to change to the directory /usr/local/share/man/man2, which is a subdirectory of the current directory, it would be possible to change by using the absolute pathname, i.e.,

cd /usr/local/share/man/man2

However, it would clearly be much less tedious to use the relative pathname, i.e.,

cd man2

On Unix-like operating systems the current directory is represented by a single dot and its parent directory (i.e., the directory that contains it) is represented by two consecutive dots. Thus, it is possible (and often convenient) to change to the parent of the current directory by using the following:

cd ..

Another convenient feature of cd is the ability for any user to return directly to its home directory by merely using a tilde as the argument. A home directory, also called a login directory, is the directory on a Unix-like operating system that serves as the repository for a user's personal files, directories and programs. It is also the directory that a user is first in after logging into the system. A tilde is a short,

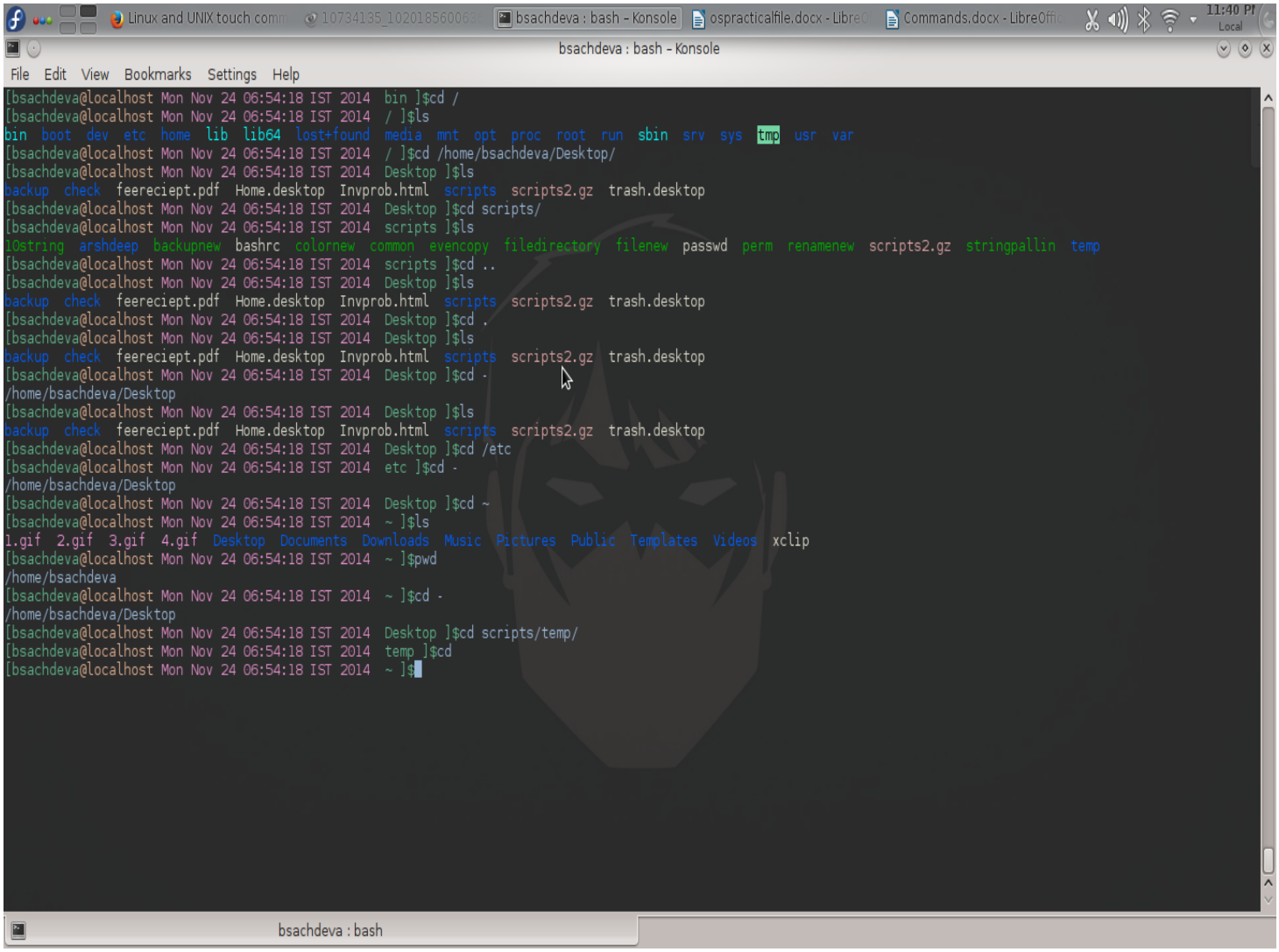
wavy, horizontal line character that represents the home directory of the current user. That is, any user can return immediately to its home directory by typing the following and then pressing the Enter key:

```
cd ~
```

This is easier than typing the full name of the user's home directory, for instance, /home/bsachdeva in the case of a user named bsachdeva. (And it is just one of the numerous shortcuts that help make the command line on Unix-like operating systems so easy to use.)

When followed by a space and then a hyphen, cd both returns the user to the previous current directory and reports on a new line the absolute pathname of that directory. This can further enhance the already convenient toggling capability of cd. Toggling is particularly convenient when at least one of the two directories has a long absolute pathname, such as /usr/local/share/man/man2.

cd has only two options, and neither of them are commonly used. The -P option instructs cd to use the physical directory structure instead of following symbolic links. The -L option forces symbolic links to be followed.



The rmdir Command

The rmdir command is used to remove empty directories in Linux and other Unix-like operating systems.Often referred to as a folder in the Macintosh and Microsoft Windows operating systems, a directory in Linux is merely a special type of file that contains a list of file names and the corresponding inodes for each file and directory that it appears (to the user) to contain. An inode is a data structure (i.e., a way of storing data so that it can be used efficiently) that stores all the information about a file except its name and its actual data.

SYNOPSIS

rmdir [option] directory_names

When used without any options, rm will delete any empty directories whose names are supplied as arguments (i.e., inputs) regardless of whether such directories have write permission or not. Thus, for example, the following command would remove two empty directories named dir1 and dir2 that are

located in the current directory (i.e., the directory in which the user is currently working):

```
rmdir dir1 dir2
```

The ability to remove only empty directories is a built-in safeguard that helps prevent the accidental loss of data. This is important because once deleted, it is extremely difficult or impossible to recover deleted data on Unix-like operating systems1.

The -p (i.e., parents) option tells rmdir to remove the parent directories of the specified directory if each successive parent directory will, in turn, become empty and if each parent directory has write permission. Thus, for example, the following would remove dir5, dir4 and dir3 if dir5 were empty, dir4 only contained dir5 and dir3 only contained dir4 (which, in turn, contained dir5):

```
rmdir -p dir3/dir4/dir5
```

This provides a nice symmetry with the -p option of the mkdir command, which is used to create directories. Thus, the above set of nested directories could be easily created with the following:

```
mkdir -p dir3/dir4/dir5
```

In contrast to the rm command, which is used to delete both files and directories, there is no -r option for rmdir. at least on the GNU version that is standard on Linux. That option allows rm to recursively delete a directory by first deleting all of its contents, beginning with those in the lowest levels of subdirectories. Thus, if a user wants to remove an entire directory structure, it is usually most efficient to use rm with its -r option rather than trying to first remove the contents of each directory, its subdirectories, etc.

Three options that rmdir shares with rm are -v (i.e., verbose), which provides additional information about what is happening, --help, which provides basic documentation about rmdir, and --version, which tells the version of rmdir that is currently in use. Some differences exist among the various versions of rmdir, so it is always wise to read the documentation for the particular system.

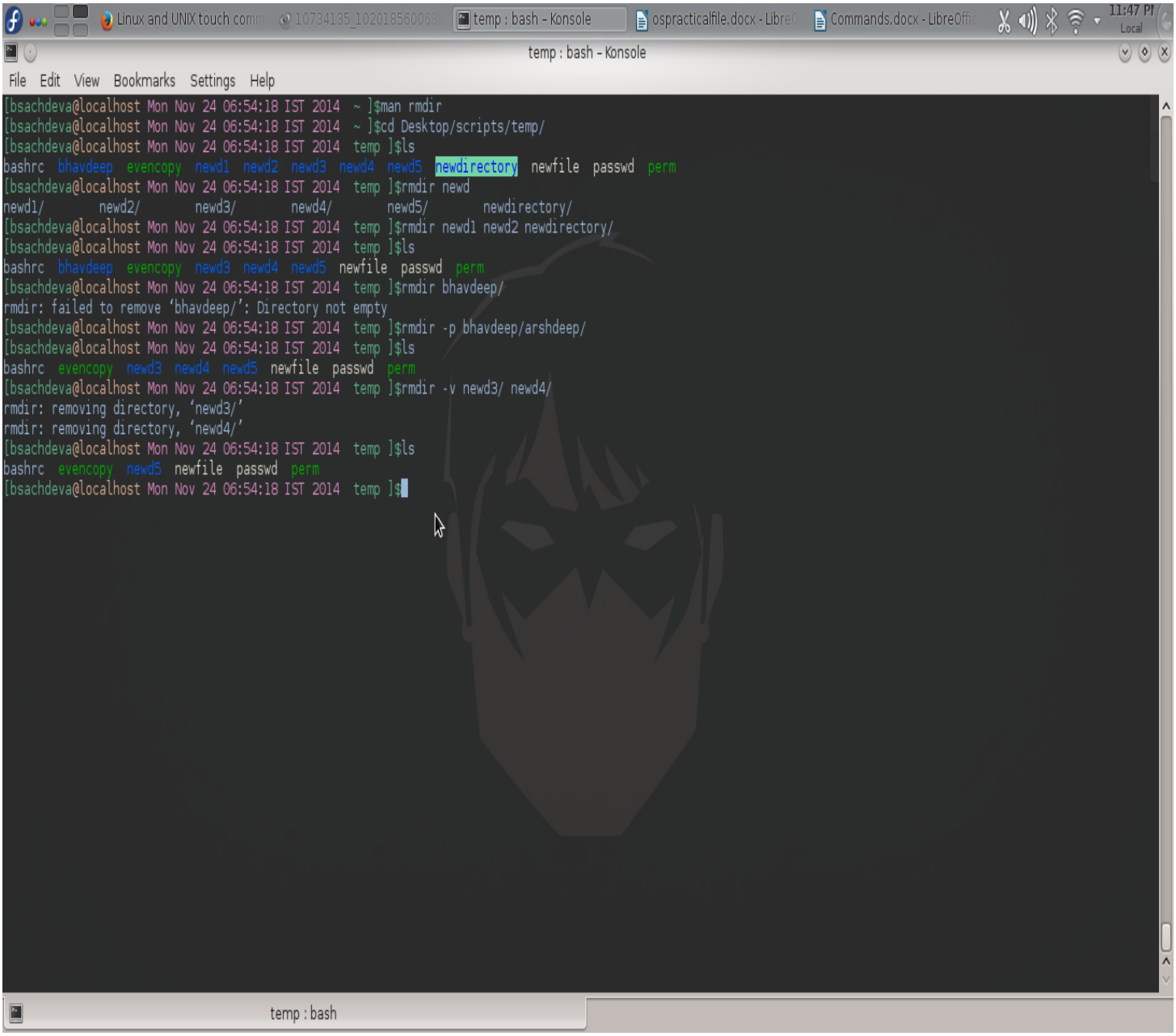
DESCRIPTION

-p, --parents

remove DIRECTORY and its ancestors; e.g., 'rmdir -p a/b/c' is similar to 'rmdir a/b/c a/b a'

-v, --verbose

output a diagnostic for every directory processed



The pwd Command

The pwd command reports the full path to the current directory.

The current directory is the directory in which a user is currently operating while using a command line interface. A command line interface is an all-text display mode and it is provided via a console (i.e., a display mode in which the entire screen is text only) or via a terminal window (i.e., a text-only window in a GUI).The full path, also called an absolute path, to a directory or file is the complete hierarchy of directories from the root directory to and including that directory or file. The root directory, which is designated by a forward slash (/), is the base directory on the file system (i.e., hierarchy of directories),

and it contains all other directories, sub-directories and files on the system. Thus, the full path for any directory or file always begins with a forward slash. pwd is one of the most basic commands in Linux and other Unix-like operating systems, along with ls, which is used to list the contents of the current directory, and cd, which is used to change the current directory.

SYNOPSIS

pwd [option]

Unlike most commands, pwd is almost always used just by itself, i.e.,

pwd

That is, it is rarely used with its options and never used with arguments (i.e., file names or other information provided as inputs). Anything that is typed on the same line after pwd, with the exception of an option, is ignored, and no error messages are returned.

As an example, if a user with the username janis is in its home directory, then the above command would typically return /home/janis/ (because, by default, all home directories are located in the directory /home). Likewise, if a user were currently working in directory /usr/share/config (which contains a number of program configuration files), then the same command would return /usr/share/config.

pwd is useful for confirming that the current directory has actually been changed to what the user intended after using cd. For example, after issuing the cd command to change the current directory from /home/janis to /usr/share/config, pwd could be used for confirmation; that is, the following sequence of commands would be issued:

cd /usr/share/config/

pwd

The standard version of pwd has a mere two options, both of which are employed only infrequently. The --help option is used as follows:

pwd --help

This option displays information about pwd, of which there is very little because it is such a simple command (i.e., it only has two options and accepts no arguments).

The other option is --version, which displays the version number, i.e.,

pwd --version

Although it is often thought of as standing for present working directory, pwd is actually an acronym for print working directory. The word print is traditional UNIX terminology for write or display, and it originated when computer output was typically printed on paper by default because CRT (cathode ray tube) display monitors were not yet widely available.

Despite its extreme simplicity, pwd remains one of the most useful and popular of the commands for

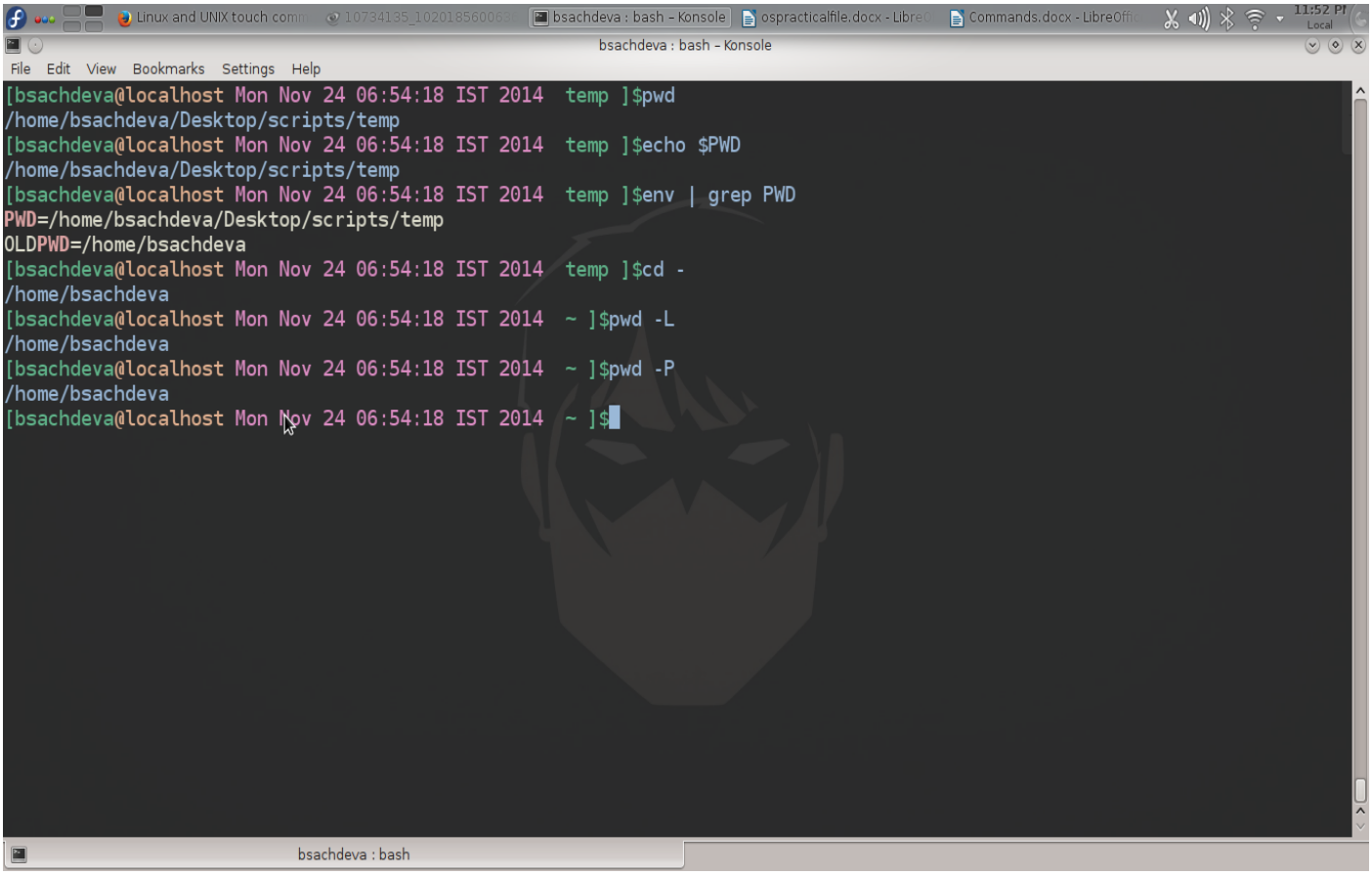
Unix-like operating systems. Actually, this simplicity is completely consistent with the Unix philosophy, which emphasizes small, specialized and modular programs rather than the large and complex programs that are favored by some other operating systems.

DESCRIPTION

-L, --logical

use PWD from environment, even if it contains symlinks

-P, --physical



avoid all symlinks

The whoami Command

The whoami command writes the user name (i.e., login name) of the owner of the current login session to standard output. Standard output is, by default, the display screen, unless redirected to a file, printer, etc. whoami is particularly useful when using shells such as ash and sh that do not show the name of the current user in the command prompt (a short text message at the start of the command line on an all-text display). It is also useful for confirming the current owner of a session after using the su (i.e.,

substitute user) command, which changes the owner of the session without the original owner having to first log out. A shell is a program that provides the traditional, text-only user interface for Unix-like operating systems. Its primary function is to read commands that are typed into a console (i.e., an all-text display mode) or terminal window (an all-text window in a GUI) and then execute (i.e., run) them.

SYNOPSIS

whoami [option]

When used without any options or redirection, as it usually is, i.e.,

whoami

and followed by pressing the ENTER key, whoami displays on the monitor screen the user name of the owner of the current session.

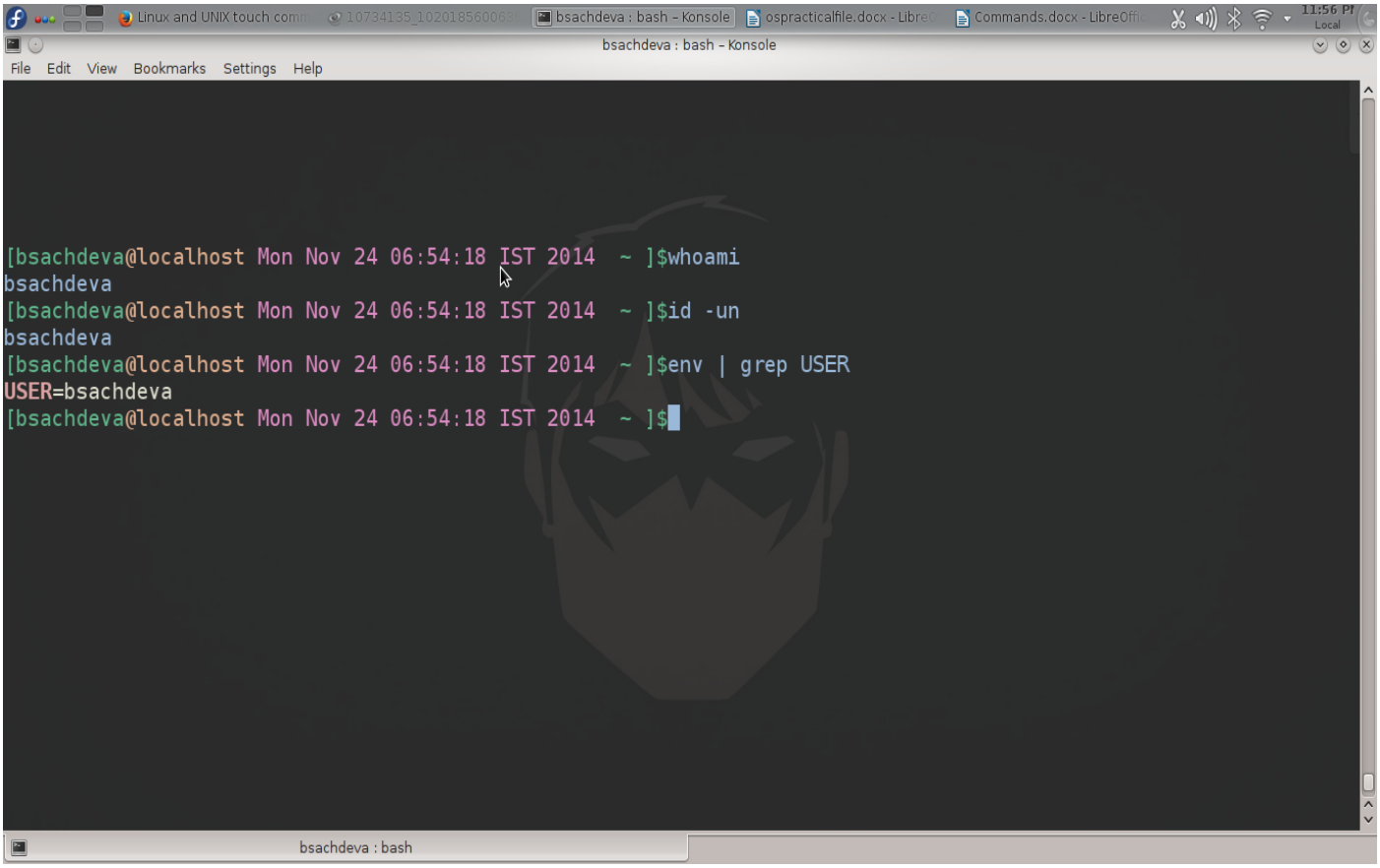
There are only two options for whoami: --help and --version. The former outputs the very brief description that is contained in the man (i.e., built-in system manual) pages, and the latter outputs the number of the version currently installed on the system.

whoami produces the same result as the id command (which by default provides more detailed information about the current user than does whoami) when id is used with its -u and -n options, i.e.,

id -un

The -u option tells id to provide only the identification for the current owner of the session, and the -n option tells it to present that identification as the user name instead of as a number.

The who command differs from whoami in that it provides a list of all users currently logged into the system as well as additional information about each of those users (including login times and terminal numbers). It also differs in that, in the event of a change in ownership of a login session through the use of the su command, it reports the original owner of the session, whereas whoami provides the user name of the effective (i.e., current) owner of the session.


bsachdeva : bash – Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$whoami
bsachdeva
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$id -un
bsachdeva
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$env | grep USER
USER=bsachdeva
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$
bsachdeva : bash

The wc Command

The wc (i.e., word count) command by default counts the number of lines, words and characters in text.

wc defines a word as a set of contiguous letters, numbers and/or symbols which are separated from other characters by one or more spaces, tabs and/or newline characters (which are generated when the RETURN key is pressed). When counting the number of characters, all characters are counted, not only letters, numbers and symbols, but also spaces, tabs and newline characters. A line is only counted if it ends with a newline character.

SYNOPSIS

wc [options] [file_name(s)]

The items in square brackets are optional. If no file names are provided, wc reads from its standard input, which by default is text entered at the keyboard.

This can be seen by typing

wc

at the command line (i.e., in the all-text mode), pressing the ENTER key to move to a new line and then typing some text on one or more lines. The command is executed (i.e., run) by pressing the ENTER key again and then pressing the CONTROL and d keys simultaneously. This causes wc to write in a new line (under the lines of text) its count of the numbers of lines, words and characters in the text.

The following command counts the number of lines, words and characters in a file named file1 that resides in the current directory (i.e., the directory in which the user is currently working) and likewise writes them, followed by the name of the file, to standard output, which is by default the display monitor:

wc file1

wc can provide its output for multiple files by listing the name of each separated by a space. For example,

wc file1 file2 file3

The numbers of lines, words and characters for each file along with its name will be displayed on a separate line and in the order that the files are listed as arguments (i.e., input files). In the case of multiple arguments such as this, wc also provides an additional line that shows the total number of lines, words and characters for all the files.

Likewise, wc can provide a count for all of the text files within a directory. This is accomplished by using the star wildcard character, which represents everything and is designated by an asterisk (*). For example, the following will display the number of lines, words and characters for each file in the current directory (which is represented by a dot) as well as totals for all files in the directory:

```
wc . *
```

wc has only a few options, the most commonly used of which restrict the information it provides. The -l option tells wc to count only the number of lines, the -w option tells it to count only the number of words, the -m option tells it to count only the number of characters and the -c option tells wc to count only the number of bytes.

Thus, for example, the following displays just the number of words in a file named file4:

```
wc -w file4
```

The following displays the number of characters in the same file:

```
wc -m file4
```

As is generally the case with commands in Unix-like operating systems, any combination of options can be used together. For example, the following would count both the numbers of lines and words in a file named file5:

```
wc -lw file5
```

Redirection can be used with wc to create more complex commands. For example, the output from the above command can be redirected using the standard output redirection operator (which is designated by a rightward pointing angle bracket) from the display screen to a file named file6 with the following:

```
wc -lw file5 > file6
```

If file6 already exists, its contents will be overwritten; if it does not exist, it will be created. The contents of file6 can be easily confirmed with a text editor or with a command such as cat, which is commonly used to read text files, i.e.,

```
cat file6
```

As another example, wc with its -l option can be used to count the total number of objects (i.e., files, links and directories) in the current directory. This is accomplished by first employing the ls command, which by default lists the contents of the current directory, and then sending its output via a pipe (designated by the vertical bar character) to wc, which counts the number of lines in the output from ls, i.e.,

```
ls | wc -l
```

wc can count the words, lines and/or characters only in text files. However, when used with its -c option, it can count the bytes in any type of file, i.e. a plain text file or a binary file. (A binary file is any file that contains at least some non-text data, such as an image file, a compressed text file or an executable program.) This can be a convenient way of determining the size of a file.

For example, the following command displays the size for each jpeg image file in the current directory as well as the total for all of them:

```
wc -c *.jpg
```

DESCRIPTION

-c, --bytes

print the byte counts

-m, --chars

print the character counts

-l, --lines

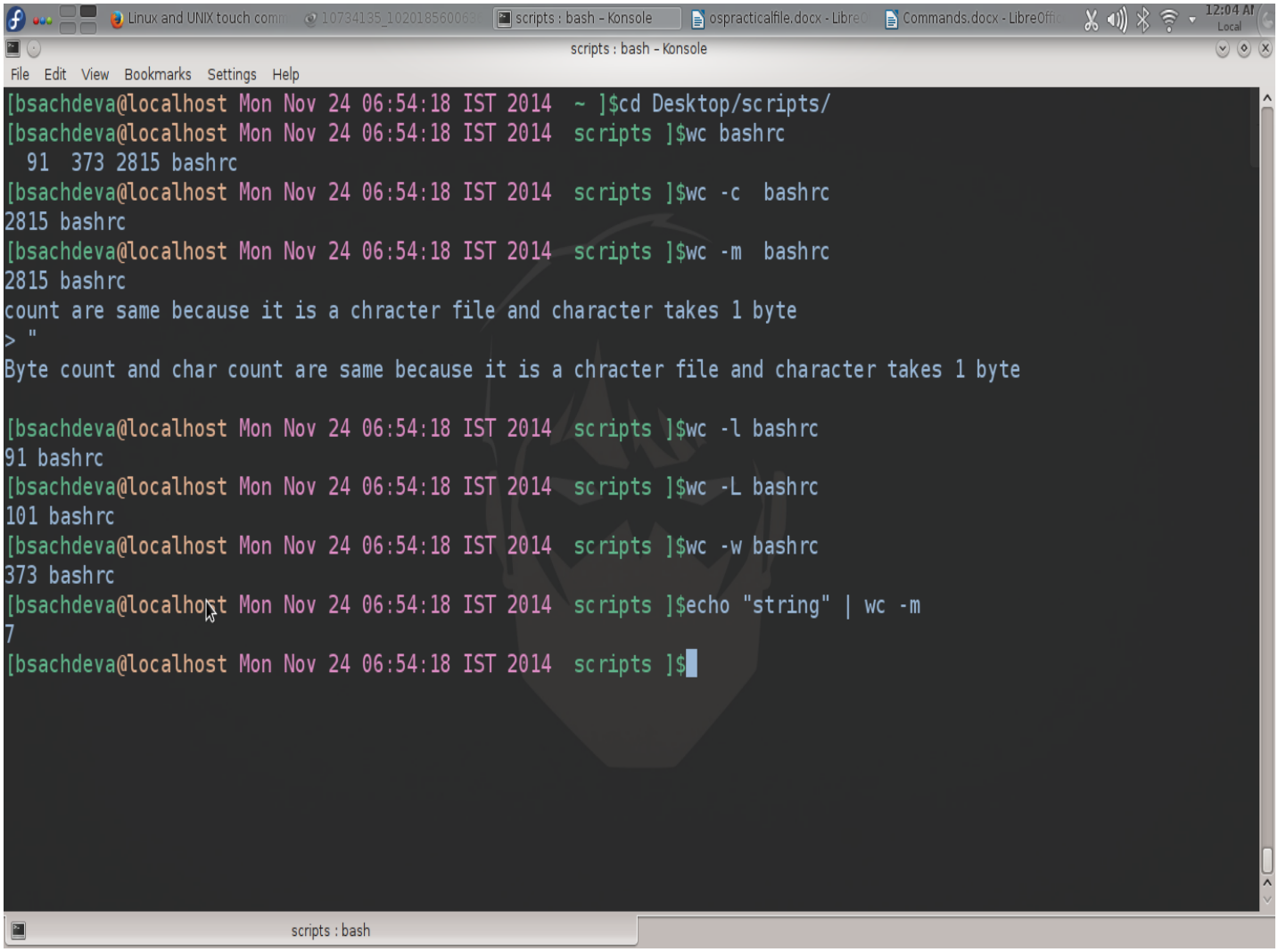
print the newline counts

-L, --max-line-length

print the length of the longest line

-w, --words

print the word counts



The head Command

The head command reads the first few lines of any text given to it as an input and writes them to standard output (which, by default, is the display screen).

SYNOPSIS

head [options] [file(s)]

The square brackets indicate that the enclosed items are optional. By default, head returns the first ten lines of each file name that is provided to it.

For example, the following will display the first ten lines of the file named aardvark in the current directory (i.e., the directory in which the user is currently working):

head aardvark

If more than one input file is provided, head will return the first ten lines from each file, precede each

set of lines by the name of the file and separate each set of lines by one vertical space. The following is an example of using head with two input files:

```
head aardvark armadillo
```

If it is desired to obtain some number of lines other than the default ten, the -n option can be used followed by an integer indicating the number of lines desired. For example, the above example could be modified to display the first 15 lines from each file:

```
head -n15 aardvark armadillo
```

-n is a very tolerant option. For example, it is not necessary for the integer to directly follow it without a space in between. Thus, the following command would produce the same result:

```
head -n  15 aardvark armadillo
```

In fact, the letter n does not even need to be used at all. Just the hyphen and the integer (with no intervening space) are sufficient to tell head how many lines to return. Thus, the following would produce the same result as the above commands:

```
head -15 aardvark armadillo
```

head can also return any desired number of bytes (i.e., a sequence of eight bits and usually long enough to represent a single character) from the start of each file rather than a desired number of lines. This is accomplished using the -c option followed by the number of bytes desired. For example, the following would display the first five bytes of each of the two files provided:

```
head -c 5 aardvark anteater
```

When head counts by bytes, it also includes the newline character, which is a non-printing (i.e, invisible) character that is designated by a backslash and the letter n (i.e., \n). Thus, for example, if there are three new, blank lines at the start of a file, they will be counted as three characters, along with the printing characters (i.e., characters that are visible on the monitor screen or on paper).

The number of bytes or lines can be followed by a multiplier suffix. That is, adding the letter b directly after the number of bytes multiplies it by 512, k multiplies it by 1024 and m multiplies it by 1048576. Thus, the following command would display the first five kilobytes of the file aardvark:

```
head -c5k aardvark
```

The -c option is less tolerant than the -n option. That is, there is no default number of bytes, and thus some integer must be supplied. Also, the letter c cannot be omitted as can the letter n, because in such case head would interpret the hyphen and integer combination as the -n option. Thus, for example, the following would produce an error message something like head: aardvark: invalid number of bytes:

```
head -c aardvark
```

If head is used without any options or arguments (i.e., file names), it will await input from the keyboard and will successively repeat (i.e., each line will appear twice) on the monitor screen each of the first ten

lines typed on the keyboard. If it were desired to repeat some number of lines other than the default ten, then the -n option would be used followed by the integer representing that number of lines (although, again, it is not necessary to include the letter n), e.g.,

```
head -n3
```

As is the case with other command line (i.e., all-text mode) programs in Linux and other Unix-like operating systems, the output from head can redirected from the display monitor to a file or printer using the output redirection operator (which is represented by a rightward-pointing angular bracket). For example, the following would copy the first 12 lines of the file Yuriko to the file December:

```
head -n 12 Yuriko > December
```

If the file named December did not yet exist, the redirection operator would create it; if it already existed, the redirection operator would overwrite it. To avoid erasing data on an existing file, the append operator (which is represented by two consecutive rightward pointing angle brackets) could be used to add the output from head to the end of a file with that name if it already existed (or otherwise create a new file with that name), i.e.,

```
head -n 12 Yuriko >> December
```

The output from other commands can be sent via a pipe (represented by the vertical bar character) to head to use as its input. For example, the following sends the output from the ls command (which by default lists the names of the files and directories in the current directory) to head, which, in turn, displays the first ten lines of the output that it receives from ls:

```
ls | head
```

This output could easily be redirected, for example to the end of a file named file1 as follows:

```
ls | head >> file1
```

It could also be piped to one or more filters for additional processing. For example, the sort filter could be used with its -r option to sort the output in reverse alphabetic order prior to appending file1:

```
ls | head | sort -r >> file1
```

The -q (i.e., quiet) option causes head to not show the file name before each set of lines in its output and to eliminate the vertical space between each set of lines when there are multiple input sources. Its opposite, the -v (i.e., verbose) option, causes head to provide the file name even if there is just a single input file.

The tail command is similar to the head command except that it reads the final lines in files rather than the first lines.

As is the case with other commands on Unix-like operating systems, additional information can be obtained about head and tail by using the man and info commands to reference the built-in documentation, for example

man head

or

info tail

DESCRIPTION

-c, --bytes=[-]K

print the first K bytes of each file; with the leading '-', print all but the last K bytes of each file

-n, --lines=[-]K

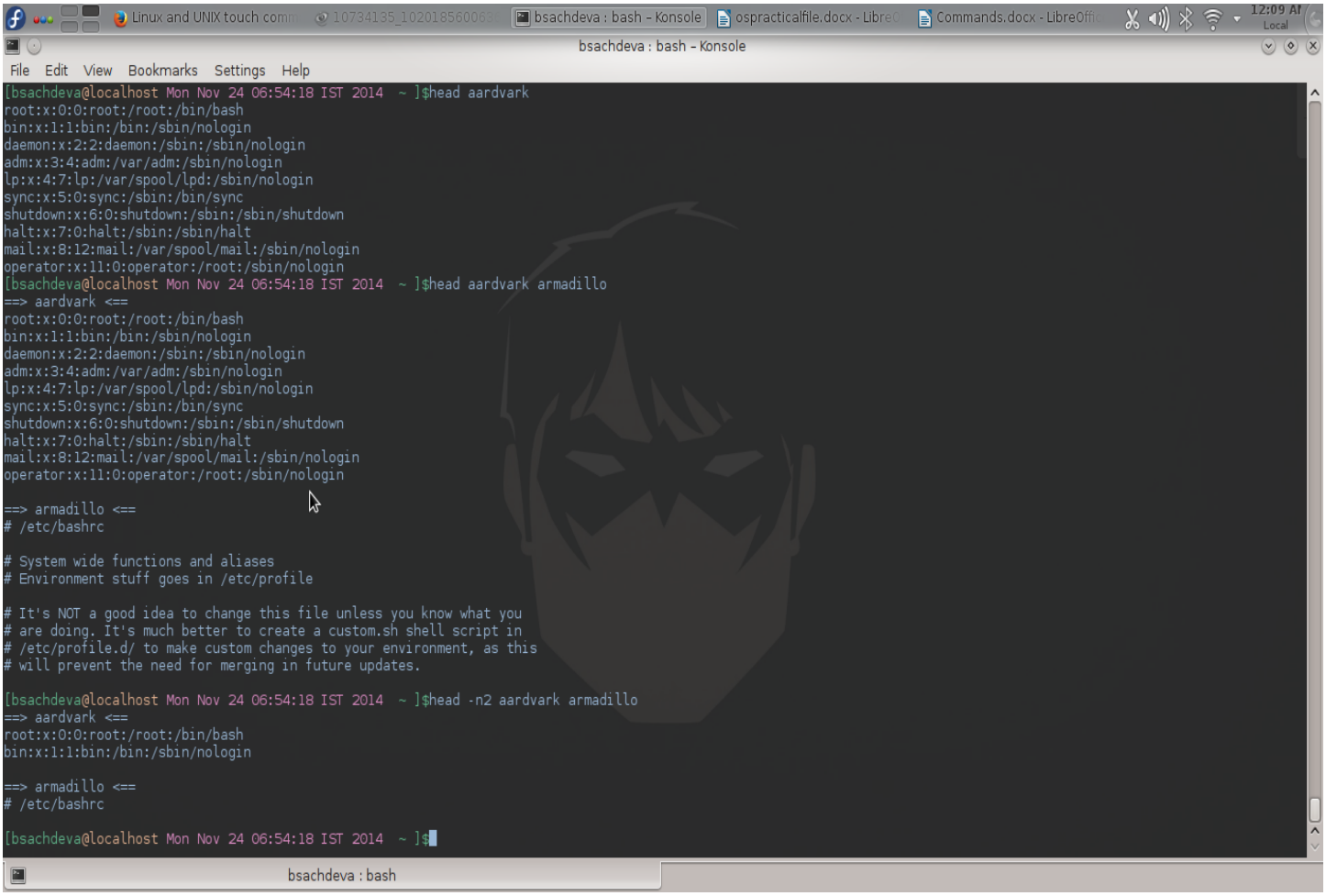
print the first K lines instead of the first 10; with the leading '-', print all but the last K lines of each file

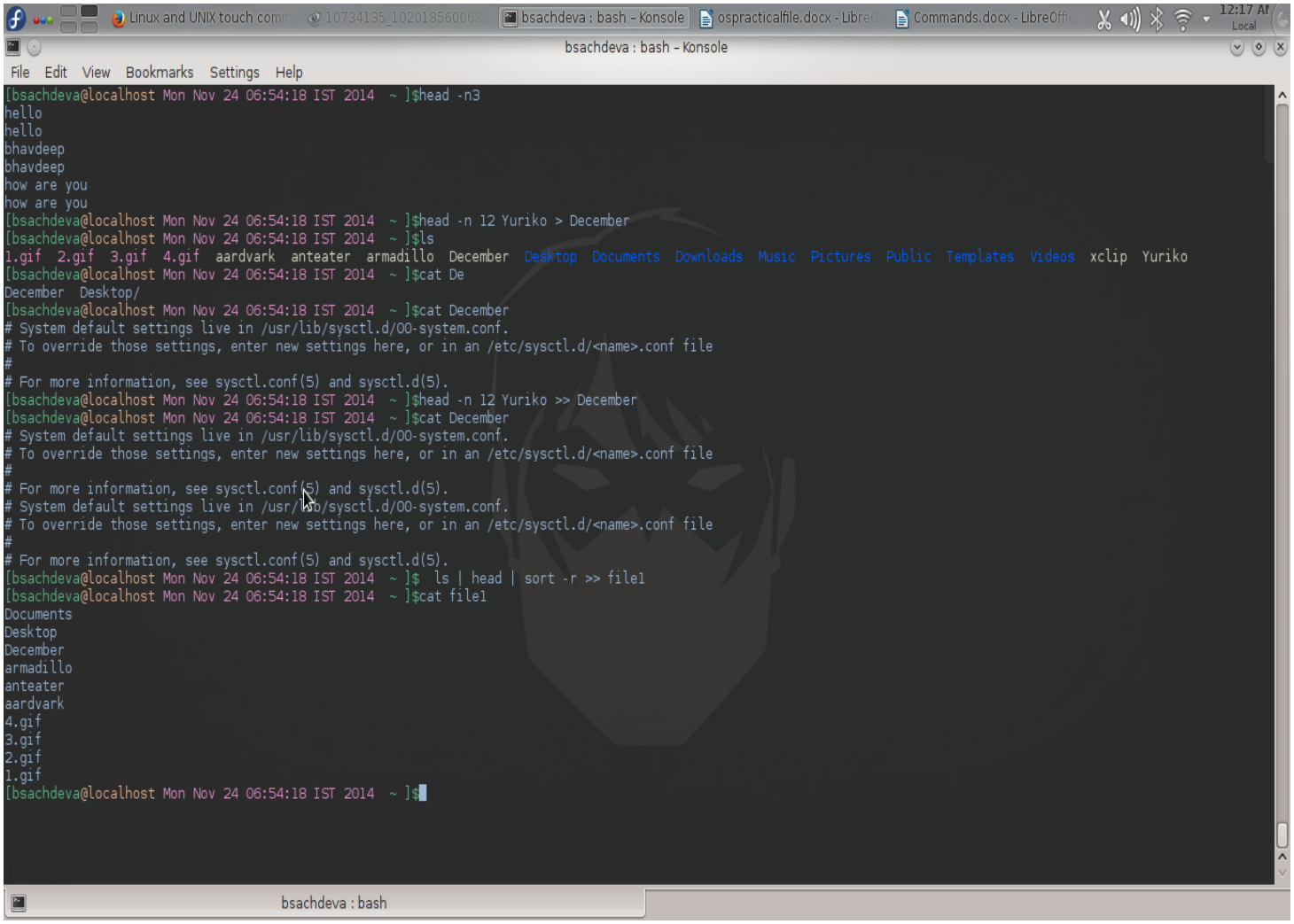
-q, --quiet, --silent

never print headers giving file names

-v, --verbose

always print headers giving file names


bsachdeva : bash – Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$head aardvark
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$head aardvark armadillo
==> aardvark <==
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
==> armadillo <==
/etc/bashrc
System wide functions and aliases
Environment stuff goes in /etc/profile
It's NOT a good idea to change this file unless you know what you
are doing. It's much better to create a custom.sh shell script in
/etc/profile.d/ to make custom changes to your environment, as this
will prevent the need for merging in future updates.
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$head -n2 aardvark armadillo
==> aardvark <==
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
==> armadillo <==
/etc/bashrc
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$
bsachdeva : bash


bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$head -n3
hello
hello
bhavdeep
bhavdeep
how are you
how are you
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$head -n 12 Yuriko > December
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ls
1.gif 2.gif 3.gif 4.gif aardvark anteater armadillo December Desktop Documents Downloads Music Pictures Public Templates Videos xclip Yuriko
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat De
December Desktop/
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat December
System default settings live in /usr/lib/sysctl.d/00-system.conf.
To override those settings, enter new settings here, or in an /etc/sysctl.d/<name>.conf file
#
For more information, see sysctl.conf(5) and sysctl.d(5).
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$head -n 12 Yuriko >> December
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat December
System default settings live in /usr/lib/sysctl.d/00-system.conf.
To override those settings, enter new settings here, or in an /etc/sysctl.d/<name>.conf file
#
For more information, see sysctl.conf(5) and sysctl.d(5).
System default settings live in /usr/lib/sysctl.d/00-system.conf.
To override those settings, enter new settings here, or in an /etc/sysctl.d/<name>.conf file
#
For more information, see sysctl.conf(5) and sysctl.d(5).
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ ls | head | sort -r >> file1
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file1
Documents
Desktop
December
armadillo
anteater
aardvark
4.gif
3.gif
2.gif
1.gif
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$
bsachdeva : bash

The tail Command

The tail command reads the final few lines of any text given to it as an input and writes them to standard output (which, by default, is the monitor screen).

SYNOPSIS

tail [options] [filenames]

The square brackets indicate that the enclosed items are optional. By default, tail returns the final ten lines of each file name that is provided to it.

For example, the following command will print (traditional Unix terminology for write) the last ten lines of the file named aardvark in the current directory (i.e., the director in which the user is currently working) to the display screen:

tail aardvark

If more than one input file is provided, tail will print the last ten lines from each file to the monitor screen. Each set of lines will be preceded by the name of the file and separated by one vertical space from other sets of lines. The following is an example of using tail with multiple input files:

tail aardvark anteater armadillo

If it is desired to print some number of lines other than the default ten, the -n option can be used followed by an integer indicating the number of lines desired. For example, to print the final 15 lines from each file in the above example, the command would be modified as follows:

tail -n15 aardvark anteater armadillo

-n is a very tolerant option. For example, it is not necessary for the integer to directly follow it without a space in between. Thus, the following command would produce the same result:

tail -n 15 aardvark anteater armadillo

In fact, the letter n does not even need to be used at all. Just the hyphen and the integer (with no intervening space) are sufficient to tell tail how many lines to print. Thus, the following would produce the same result as the above commands:

tail -15 aardvark anteater armadillo

tail can also print any desired number of bytes (i.e., a sequence of eight bits and usually long enough to represent a single character) from the end of each file rather than a desired number of lines. This is accomplished using the -c option followed by the number of bytes desired. For example, to view the final five bytes of each of the two files aardvark and anteater, the following command would be used:

tail -c 5 aardvark anteater

When tail counts by bytes, it also includes the newline character, which is a non-printing (i.e, invisible)

character that is designated by a backward slash and the letter n (i.e., \n). Thus, for example, if there are three new, blank lines at the end of a file, they will be counted as three characters, along with the printing characters (i.e., characters that are visible on the monitor screen or paper).

The number of bytes or lines can be followed by a multiplier suffix. That is, adding the letter b directly after the number of bytes multiplies it by 512, k multiplies it by 1024 and m multiplies it by 1048576. Thus, the following command would print the last five kilobytes of the file aardvark:

```
tail -c5k aardvark
```

The -c option is less tolerant than the -n option. That is, there is no default number of bytes, and thus some integer must be supplied. Also, the letter c cannot be omitted as can the letter n, because in such case tail would interpret the hyphen and integer combination as the -n option. Thus, for example, the following would produce an error message something like tail: aardvark: invalid number of bytes:

```
tail -c aardvark
```

If tail is used without any options or arguments (i.e., inputs), it will await input from the keyboard and will successively repeat (i.e., each line will appear twice) on the monitor screen each of the final ten lines typed on the keyboard. If it were desired to repeat some number of lines other than the default ten, then the -n option would be used followed by the integer representing that number of lines (although, again, it is not necessary to include the letter n), e.g.,

```
tail -n3
```

As is the case with other command line (i.e., all-text mode) programs in Unix-like operating systems, the output of tail can be redirected from the monitor to a file or printer using the redirection operator (which is represented by a rightward pointing angular bracket). For example, the following would write the final 12 lines of the file Yuriko to the file December:

```
tail -n 12 Yuriko > December
```

If the file named December did not yet exist, the redirection operator would create it; if it already existed, the redirection operator would overwrite it. To avoid erasing data on an existing file, the append operator (which is represented by two rightward pointing angular brackets) could be used to add the output from tail to the end of a file with that name if it already existed (or otherwise create a new file with that name), i.e.,

```
tail -n 12 Yuriko >> December
```

The output from other commands can be piped (i.e., sent) to tail to use as its input. For example, the following sends the output from the ls command (which by default lists the names of the files and directories in the current directory) to tail, which, in turn, prints the final ten lines of the output that it receives from ls to the monitor screen:

```
ls | tail
```

This output could easily be redirected, for example to a file named last_filenames as follows:

```
ls | tail >>last_filenames
```

It could also be piped to one or more filters for additional processing. For example, the sort filter could be used with its -r option to sort the output in reverse alphabetic order prior to writing to a file:

```
ls | tail | sort -r >> last_filenames
```

The -q (i.e., quiet) option causes tail to not print the file name before each set of lines and to eliminate the vertical space between each set of lines when there are multiple input sources. The -v (i.e., verbose) option causes tail to print the file name even if there is just a single input file.

Tail could be viewed as a counterpart of the head command, which always starts reading from the beginning of files and which can continue until any specified distance from the beginning. However, there are a few differences. Perhaps the most useful of these is that tail is somewhat more flexible in that, in addition to being able to start reading any specified distance from the end of a file, it can also start at any specified distance from the beginning of a file.

Tail can be instructed to begin printing from some number of lines or bytes from the start of a file by preceding the number with a plus sign instead of a minus sign. For example, the following would print each of the designated files to the display monitor beginning with the seventh line and until the end:

```
tail +7 aardvark anteater armadillo
```

The c option could be used to tell tail to print each of the designated files beginning with the seventh byte instead of the seventh line:

```
tail +7c aardvark anteater armadillo
```

A particularly common application for tail is examining the most recent entries in log files. This is because the newest entries are appended to the ends of such files, which tail excels in showing. As log files can be a rather long, this can eliminate a lot of scrolling that would be necessary if some other command were used to read them. For example, the most recent entries to the log /var/log/messages can easily be viewed by using the following:

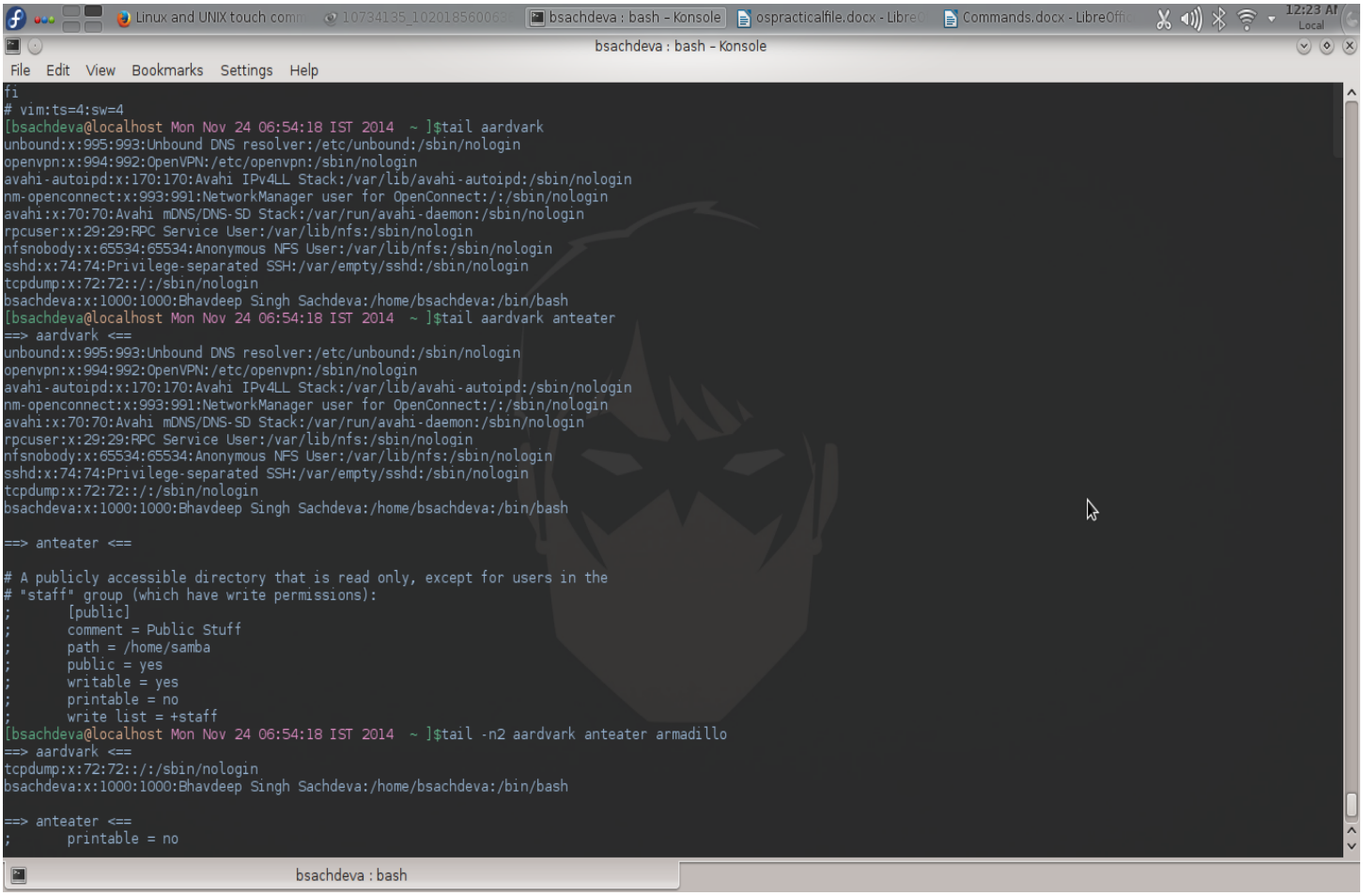
```
tail /var/log/messages
```

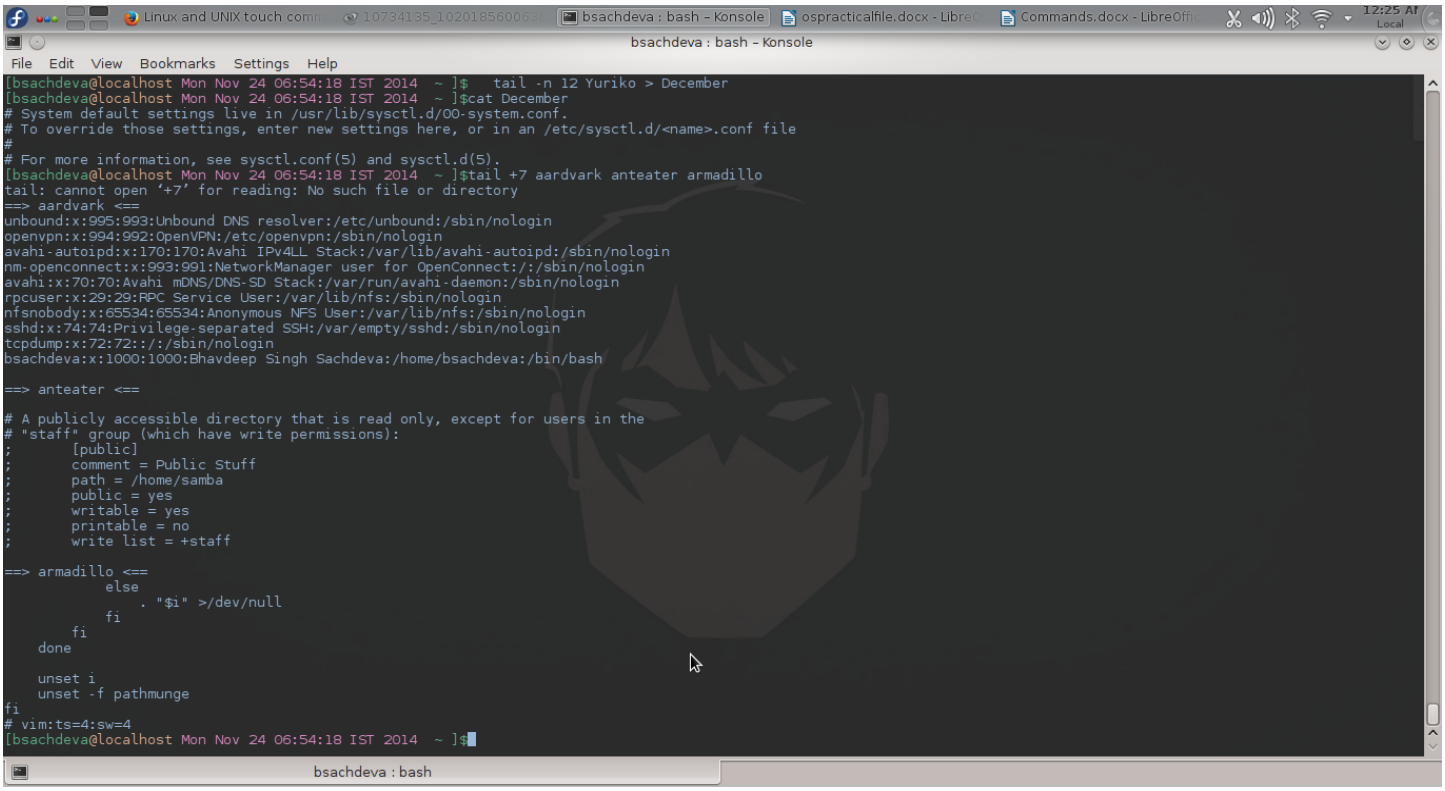
As is the case with other programs on Unix-like operating systems, additional information, including variations on specific versions, can be obtained about the tail and head commands by consulting the built-in manual and information pages using commands such as:

```
man tail
```

or

```
info head
```


bsachdeva : bash – Konsole
File Edit View Bookmarks Settings Help
fi
vim:ts=4:sw=4
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$tail aardvark
unbound:x:995:993:Unbound DNS resolver:/etc/unbound:/sbin/nologin
openvpn:x:994:992:OpenVPN:/etc/openvpn:/sbin/nologin
avahi-autoipd:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/sbin/nologin
nm-openconnect:x:993:991:NetworkManager user for OpenConnect:/:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
bsachdeva:x:1000:1000:Bhavdeep Singh Sachdeva:/home/bsachdeva:/bin/bash
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$tail aardvark anteater
==> aardvark <==
unbound:x:995:993:Unbound DNS resolver:/etc/unbound:/sbin/nologin
openvpn:x:994:992:OpenVPN:/etc/openvpn:/sbin/nologin
avahi-autoipd:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/sbin/nologin
nm-openconnect:x:993:991:NetworkManager user for OpenConnect:/:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
bsachdeva:x:1000:1000:Bhavdeep Singh Sachdeva:/home/bsachdeva:/bin/bash
==> anteater <==
A publicly accessible directory that is read only, except for users in the
"staff" group (which have write permissions):
; [public]
; comment = Public Stuff
; path = /home/samba
; public = yes
; writable = yes
; printable = no
; write list = +staff
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$tail -n2 aardvark anteater armadillo
==> aardvark <==
tcpdump:x:72:72::/:/sbin/nologin
bsachdeva:x:1000:1000:Bhavdeep Singh Sachdeva:/home/bsachdeva:/bin/bash
==> anteater <==
; printable = no
bsachdeva : bash


bsachdeva : bash – Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$ tail -n 12 Yuriko > December
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat December
System default settings live in /usr/lib/sysctl.d/00-system.conf.
To override those settings, enter new settings here, or in an /etc/sysctl.d/<name>.conf file
#
For more information, see sysctl.conf(5) and sysctl.d(5).
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$tail +7 aardvark anteater armadillo
tail: cannot open '+7' for reading: No such file or directory
==> aardvark <==
unbound:x:995:993:Unbound DNS resolver:/etc/unbound:/sbin/nologin
openvpn:x:994:992:OpenVPN:/etc/openvpn:/sbin/nologin
avahi-autoipd:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/sbin/nologin
nm-openconnect:x:993:991:NetworkManager user for OpenConnect:/:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
bsachdeva:x:1000:1000:Bhavdeep Singh Sachdeva:/home/bsachdeva:/bin/bash
==> anteater <==
A publicly accessible directory that is read only, except for users in the
"staff" group (which have write permissions):
; [public]
; comment = Public Stuff
; path = /home/samba
; public = yes
; writable = yes
; printable = no
; write list = +staff
==> armadillo <==
else
. "$i" >/dev/null
fi
fi
done
unset i
unset -f pathmunge
fi
vim:ts=4:sw=4
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$
bsachdeva : bash

The cp Command

The cp command is used to copy files and directories. The copies become independent of the originals (i.e., a subsequent change in one will not affect the other).

SYNOPSIS

cp [options] name new_name

As a safety precaution, by default cp only copies files and not directories. If a file with the same name as that assigned to the copy of a file (or a directory with the same name as that assigned to the copy of a directory) already exists, it will be overwritten (i.e., its contents will be lost). However, the owner, group and permissions for the copy become the same as those of the file with the same name that it replaced. The last access time of the source file and the last modification time of the new file are set to the time the copying was performed.

When a copy is made of a file or directory, the copy must have a different name than the original if it is to be placed in the same directory as the original. However, the copy can have the same name if it is made in a different directory. Thus, for example, a file in the current directory (i.e., the directory in which the user is currently working) named file1 could be copied with the same name into another directory, such as into /home/john/, as follows:

cp file1 /home/bsachdeva/file1

Any number of files can be simultaneously copied into another directory by listing their names followed by the name of the directory. cp is an intelligent command and knows to do this when only the final argument (i.e., piece of input data) is a directory. The files copied into the directory will all have the same names as the originals. Thus, for example, the following would copy the files named file2, file3 and file4 into a directory named dir1:

cp file2 file3 file4 dir1

The -r (i.e., recursive) option, which can also be written with an upper case R, allows directories including all of their contents to be copied. (Directories are not copied by default in order to make it more difficult for users to accidentally overwrite existing directories which have the same name as that assigned to the copy being made and which might contain critical directory structures or important data.) Thus, for example, the following command would make a copy of an existing directory called dir2, inclusive of all it contents (i.e., files, subdirectories, their subdirectories, etc.), called dir3:

cp -r dir2 dir3

The -i (i.e., interactive) option prompts the user in the event that any name assigned to a copy is already in use by another file and that file would thus be overwritten. Entering the letter y (either lower case or upper case) in response to the prompt causes the command to continue; any other answer prevents the command from overwriting the file. Thus, for example, if it is desired to make a copy of a directory called dir4 and call it dir5 and if a directory named dir4 already exists, the following would prompt the

user prior to replacing any files with identical names in the latter directory:

```
cp -ri dir4 dir5
```

The -a option preserves as much of the structure and attributes of the original directory and its contents as possible in the new directory and is thus useful for creating archives. It is similar to the -r option in that it copies directories recursively; however, it also never follows symbolic links. It is equivalent to the -rdp combination of options.

All the files in a directory can be copied to another directory by using the star wildcard. The star character represents any single character or any combination of characters. Thus, for example, the following would copy all of the files in a directory named dir6 into another existing directory called dir7:

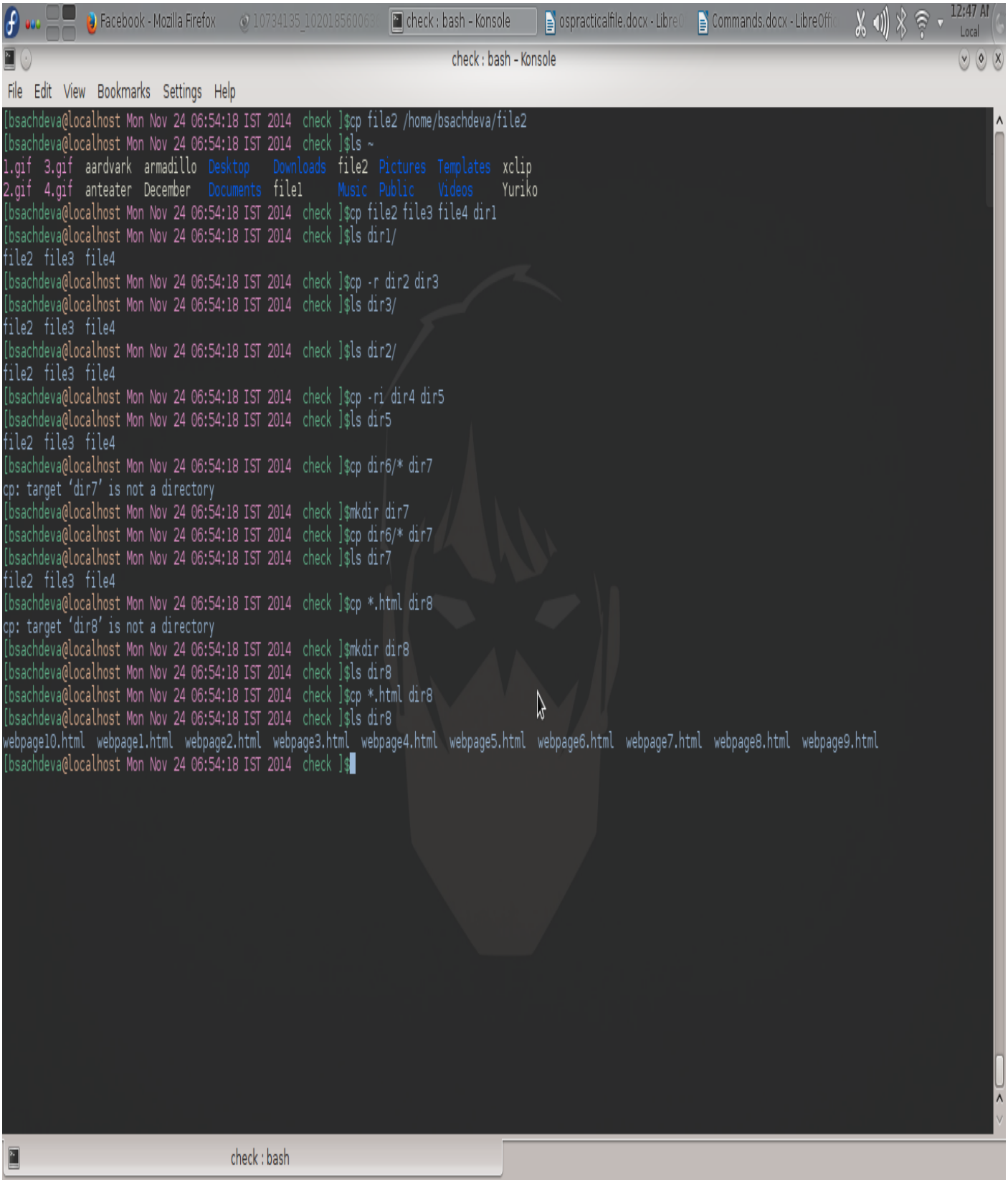
```
cp dir6/* dir7
```

cp can also be used with the star wildcard or other pattern matching characters to selectively copy files and directories. For example, to copy all of the files in the current directory that have the filename extension .html into another existing directory called dir8, the following would be used:

```
cp *.html dir8
```

In this case, the star wildcard represents anything whose name ends with the .html extension.

Among the other options for cp are -b, which makes backup copies of each destination file, -f (i.e., force), which removes destination files that cannot be opened and tries again, -s, which makes symbolic links instead of copying, -u (i.e., update), which copies only if the source file is newer than the destination file or if the destination file is missing, -v (i.e., verbose), which makes brief comments about what is going on, and -x, which tells cp to stay on the same filesystem.


Facebook - Mozilla Firefox
check : bash - Konsole
ospracticalfile.docx - Libre
Commands.docx - LibreOffi
12:47 AM
Local
check : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$cp file2 /home/bsachdeva/file2
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$ls ~
1.gif 3.gif aardvark armadillo Desktop Downloads file2 Pictures Templates xclip
2.gif 4.gif anteater December Documents file1 Music Public Videos Yuriko
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$cp file2 file3 file4 dir1
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$ls dir1/
file2 file3 file4
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$cp -r dir2 dir3
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$ls dir3/
file2 file3 file4
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$ls dir2/
file2 file3 file4
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$cp -ri dir4 dir5
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$ls dir5
file2 file3 file4
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$cp dir6/* dir7
cp: target 'dir7' is not a directory
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$mkdir dir7
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$cp dir6/* dir7
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$ls dir7
file2 file3 file4
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$cp *.html dir8
cp: target 'dir8' is not a directory
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$mkdir dir8
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$ls dir8
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$cp *.html dir8
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$ls dir8
webpage10.html webpage1.html webpage2.html webpage3.html webpage4.html webpage5.html webpage6.html webpage7.html webpage8.html webpage9.html
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$
check : bash

The mv Command

The mv command is used to rename and move files and directories.

SYNOPSIS

mv [options] argument(s)

The arguments are names of files and directories. If two file names are provided as arguments, mv renames the first as the second. If a list of arguments is provided and the final argument in the sequence is the name of an existing directory, mv moves all of the other items into that directory. If the final argument is not an existing directory and more than two arguments are provided, an error message is returned.

mv's syntax can also be expressed in a less general form as:

mv [options] source target

If the target file is located in the same directory as the source file, then the source file can only be renamed. If both are in different directories, then the source file is moved to the directory named in the target argument, in which it can keep its original name or be assigned a new name. If the target is a directory, then the source file or directory is moved into that directory and retains its original name.

Thus, for example, the following would rename a file called file1 to file2, while keeping it in the current directory (i.e., the directory in which the user is currently working):

mv file1 file2

The following would move a file named file3, without changing its name, from the current directory to an existing subdirectory of the current directory named dir1:

mv file3 dir1/file3

mv can be used to move any number of files and directories simultaneously. For example, the following command moves all files and directories, including all the contents of those directories, from the current directory to the directory /home/alice/new/:

mv * /home/alice/new/

The asterisk is a wildcard character that represents any string (i.e., sequence of characters). Thus, in the above example it represents the name of every file and directory in the current directory.

mv makes it as easy to move a file or directory up the hierarchy of directories (i.e., closer to the root directory) as down it. For example, the following would move a file named file4, which is currently located in the sub-subdirectory dir/dir/ of the user's home directory, to the top level in the user's home directory:

mv dir/dir/file4 ~

The root directory is the directory that contains all other directories on a Unix-like operating system and which is at the top of the hierarchy of directories. A user's home directory is the directory in which a user finds itself by default after logging into the system and which can be represented by the tilde (wavy horizontal line character).

By default, mv does not provide any confirmation on the display screen if its action is completed without problems. This is consistent with the rule of silence tenet of the Unix philosophy.

Thus it is wise for users new to Unix-like operating systems to always use the -i option, which makes mv interactive in the situation in which files and/or directories with the same name already exist in the destination directory. For example, the above command would be made interactive as follows:

```
mv -i * /home/alice/new/
```

Among mv's few other options are -b, which tells it to make a backup copy of each file that would otherwise be overwritten or removed, and -v, which tells it to be verbose and display the name of each file before moving it. Detailed information (including all options) about mv can be obtained by using its --help option, and information about the current version can be obtained by using its --version option.

DESCRIPTION

--backup[=CONTROL]

make a backup of each existing destination file

-b like --backup but does not accept an argument

-f, --force

do not prompt before overwriting

-i, --interactive

prompt before overwrite

-n, --no-clobber

do not overwrite an existing file

-t, --target-directory=DIRECTORY

move all SOURCE arguments into DIRECTORY

-T, --no-target-directory

treat DEST as a normal file

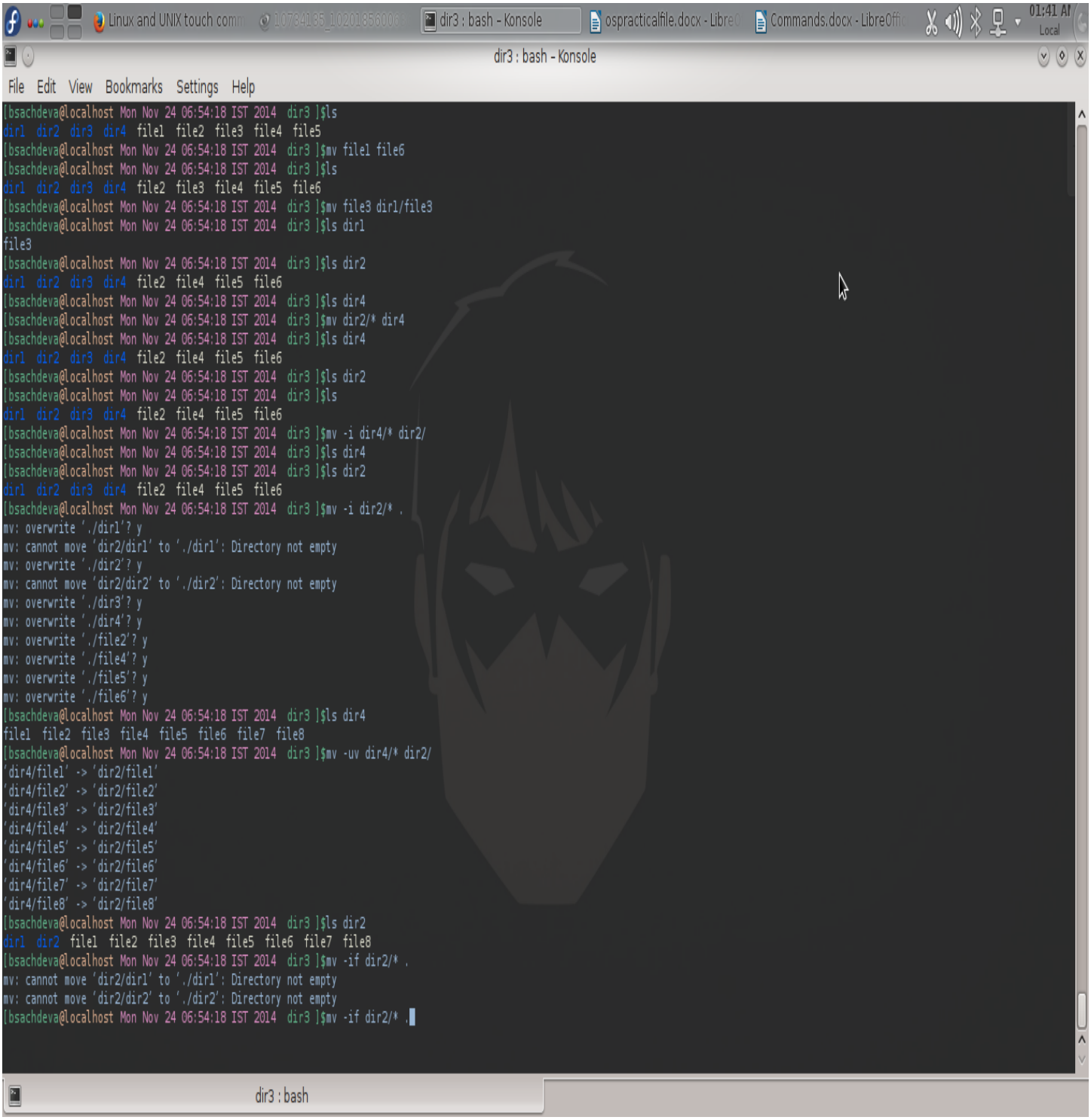
-u, --update

move only when the SOURCE file is newer than the destination file or when the destination

file is missing

-v, --verbose

explain what is being done



The cal command

The cal command isused to display a calendar or print a calendar. If no arguments are specified, the current month is displayed. The cal command is part of the util-linux package.

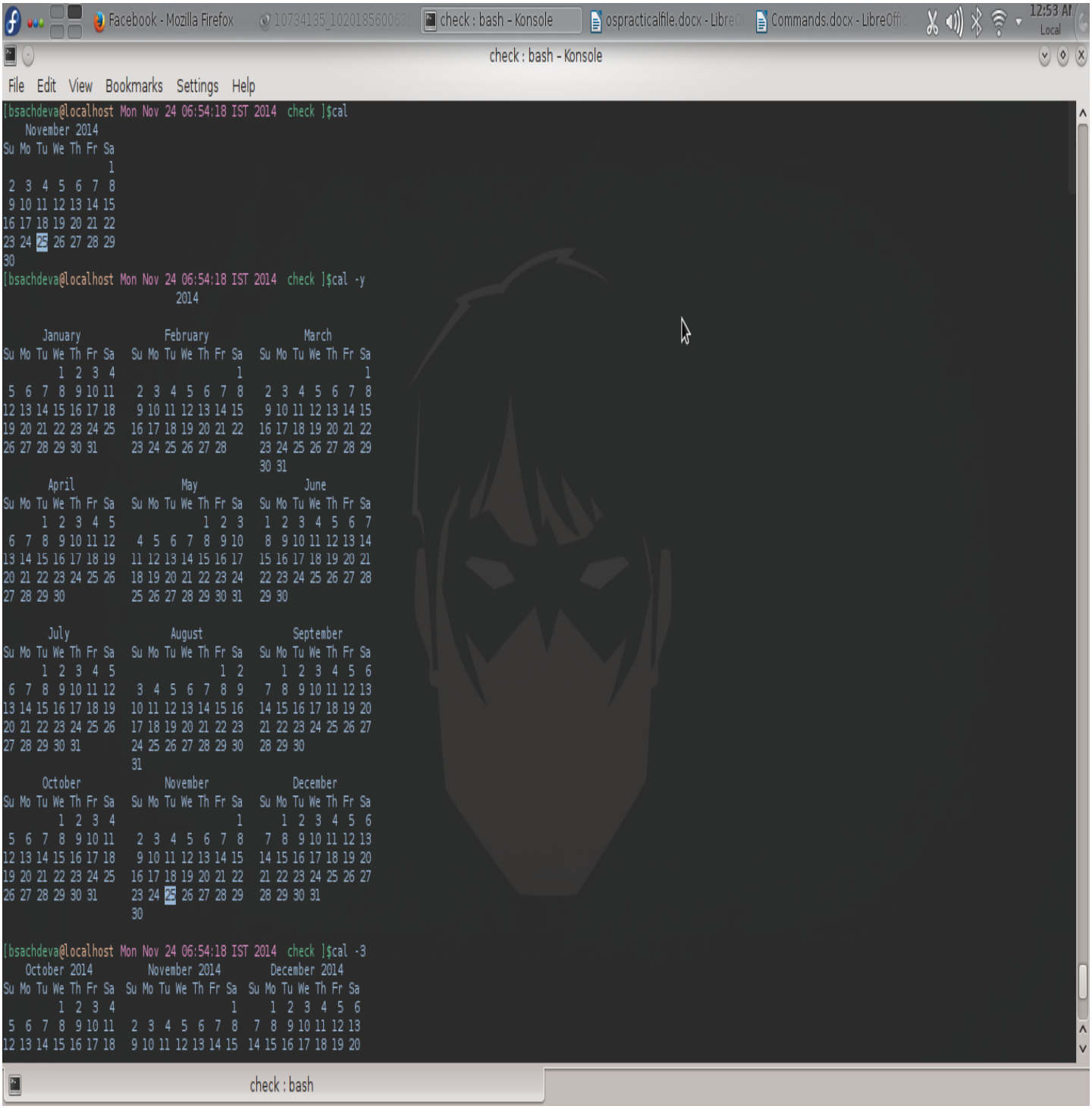
A single parameter specifies the year to be displayed; note the year must be fully specified: cal 89 will not display a calendar for 1989.

Two parameters denote the month (1 - 12) and year.

Three parameters denote the day (1-31), month and year, and the day will be highlighted if the calendar is displayed on a terminal. If no parameters are specified, the current month's calendar is displayed.

A year starts on Jan 1. The first day of the week is determined by the locale.

The Gregorian Reformation is assumed to have occurred in 1752 on the 3rd of September. By this time, most countries had recognized the reformation (although a few did not recognize it until the early 1900's). Ten days following that date were eliminated by the reformation, so the calendar for that month is a bit unusual.


Facebook - Mozilla Firefox
check : bash - Konsole
ospracticalfile.docx - LibreO
Commands.docx - LibreOffi
12:53 AM
Local
check : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$cal
November 2014
Su Mo Tu We Th Fr Sa
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$cal -y
2014
January
Su Mo Tu We Th Fr Sa
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
February
Su Mo Tu We Th Fr Sa
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28
March
Su Mo Tu We Th Fr Sa
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
April
Su Mo Tu We Th Fr Sa
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
May
Su Mo Tu We Th Fr Sa
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
June
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
July
Su Mo Tu We Th Fr Sa
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
August
Su Mo Tu We Th Fr Sa
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
September
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
October
Su Mo Tu We Th Fr Sa
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
November
Su Mo Tu We Th Fr Sa
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
December
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 check]$cal -3
October 2014
Su Mo Tu We Th Fr Sa
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
November 2014
Su Mo Tu We Th Fr Sa
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
December 2014
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
check : bash

The sort command

Sort command is helpful to sort/order lines in text files. You can sort the data in text file and display the output on the screen, or redirect it to a file. Based on your requirement, sort provides several command line options for sorting data in a text file.

SYNOPSIS

```
sort [-options]
```

For example, here is a test file:

```
$ cat test
zzz
sss
qqq
aaa
BBB
ddd
AAA
```

And, here is what you get when sort command is executed on this file without any option. It sorts lines in test file and displays sorted output.

```
$ sort test
aaa
AAA
BBB
ddd
qqq
sss
zzz
```

If we want to sort on numeric value, then we can use *-n* or *–numeric-sort* option.

Create the following test file for this example:

```
$ cat test
22 zzz
33 sss
11 qqq
77 aaa
55 BBB
```

The following sort command sorts lines in test file on numeric value in first word of line and displays sorted output.

```
$ sort -n test
11 qqq
22 zzz
33 sss
55 BBB
77 aaa
```

If we want to sort on human readable numbers (e.g., 2K 1M 1G), then we can use *-h* or *–human-numeric-sort* option.

Create the following test file for this example:

```
$ cat test
2K
2G
1K
6T
1T
1G
2M
```

The following sort command sorts human readable numbers (i.e 1K = 1 Thousand, 1M = 1 Million, 1G = 1 Giga, 1T = 1 Tera) in test file and displays sorted output.

```
$ sort -h test
1K
2K
2M
1G
2G
1T
6T
```

If we want to sort in the order of months of year, then we can use *-M* or *–month-sort* option.

Create the following test file for this example:

```
$ cat test
sept
aug
jan
oct
apr
feb
mar11
```

The following sort command sorts lines in test file as per month order. Note, lines in file should contain at least 3 character name of month name at start of line (e.g. jan, feb, mar). If we will give, ja for January or au for August, then sort command would not consider it as month name.

```
$ sort -M test
jan
feb
mar11
apr
aug
sept
oct
```

If we want to check data in text file is sorted or not, then we can use *-c* or *–check, –check=diagnose-first* option.

Create the following test file for this example:

```
$ cat test
2
5
1
6
```

The following sort command checks whether text file data is sorted or not. If it is not, then it shows first occurrence with line number and disordered value.

```
$ sort -c test
sort: test:3: disorder: 1
```

If we want to get sorted output in reverse order, then we can use *-r* or *–reverse* option. If file contains duplicate lines, then to get unique lines in sorted output, "-u" option can be used.

Create the following test file for this example:

```
$ cat test
5
2
2
1
4
4
```

The following sort command sorts lines in test file in reverse order and displays sorted output.

```
$ sort -r test
5
4
4
2
2
1
```

The following sort command sorts lines in test file in reverse order and removes duplicate lines from sorted output.

```
$ sort -r -u test
5
4
2
1
```

If we want to sort on the column or word position in lines of text file, then "-k" option can be used. If we each word in each line of file is separated by delimiter except 'space', then we can specify delimiter using "-t" option. We can get sorted output in any specified output file (using "-o" option) instead of

displaying output on standard output.

Create the following test file for this example:

```
$ cat test
aa aa zz
aa aa ff
aa aa tt
aa aa kk
```

The following sort command sorts lines in test file on the 3rd word of each line and displays sorted output.

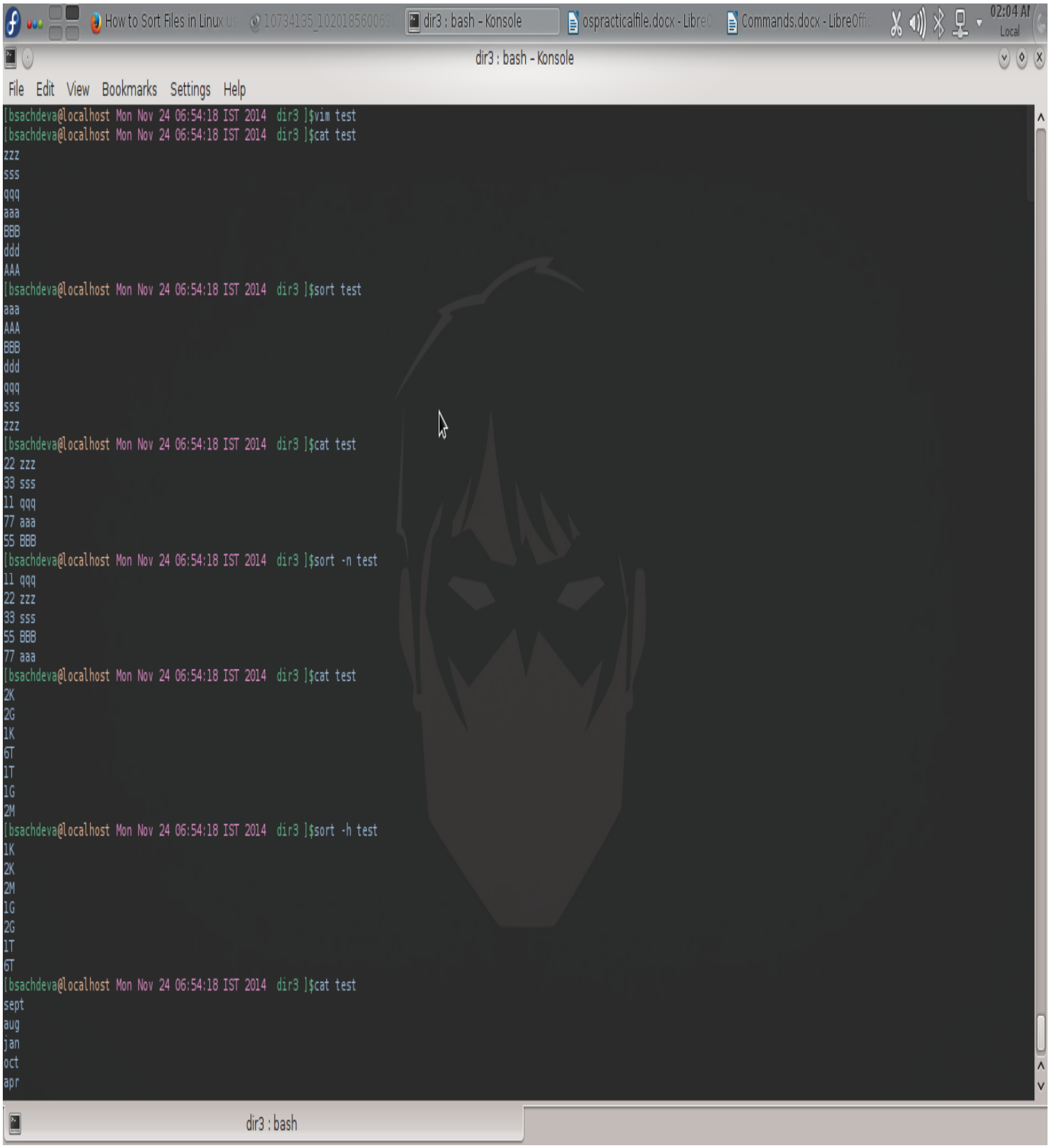
```
$ sort -k3 test
aa aa ff
aa aa kk
aa aa tt
aa aa zz

$ cat test
aa|5a|zz
aa|2a|ff
aa|1a|tt
aa|3a|kk
```

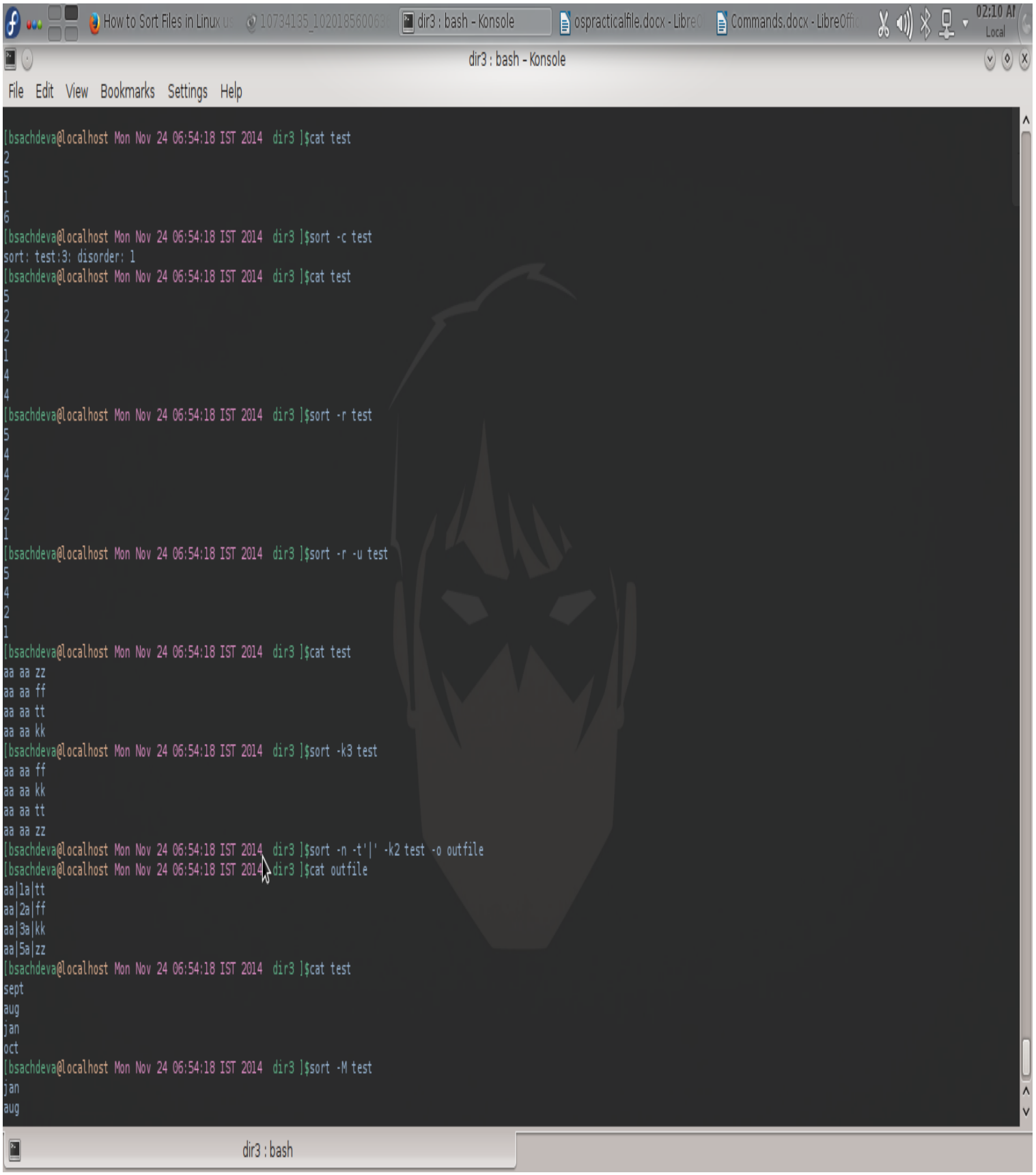
Here, several options are used altogether. In test file, words in each line are separated by delimiter '|'. It sorts lines in test file on the 2nd word of each line on the basis of numeric value and stores sorted output into specified output file.

```
$ sort -n -t'|' -k2 test -o outfile
```

The contents of output file are shown below.

```
$ cat outfile
aa|1a|tt
aa|2a|ff
aa|3a|kk
aa|5a|zz
```


How to Sort Files in Linux
dir3 : bash - Konsole
ospracticalfile.docx - LibreO
Commands.docx - LibreOffi
02:04 AM
Local
dir3 : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$vim test
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$cat test
zzz
sss
qqq
aaa
BBB
ddd
AAA
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$sort test
aaa
AAA
BBB
ddd
qqq
sss
zzz
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$cat test
22 zzz
33 sss
11 qqq
77 aaa
55 BBB
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$sort -n test
11 qqq
22 zzz
33 sss
55 BBB
77 aaa
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$cat test
2K
2G
1K
6T
1T
1G
2M
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$sort -h test
1K
2K
2M
1G
2G
1T
6T
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$cat test
sept
aug
jan
oct
apr
dir3 : bash


How to Sort Files in Linux
dir3 : bash - Konsole
ospracticalfile.docx - LibreO
Commands.docx - LibreOffic
02:10 AM
Local
dir3 : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$cat test
2
5
1
6
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$sort -c test
sort: test:3: disorder: 1
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$cat test
5
2
2
1
4
4
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$sort -r test
5
4
4
2
2
1
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$sort -r -u test
5
4
2
1
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$cat test
aa aa zz
aa aa ff
aa aa tt
aa aa kk
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$sort -k3 test
aa aa ff
aa aa kk
aa aa tt
aa aa zz
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$sort -n -t'|' -k2 test -o outfile
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$cat outfile
aa|1a|tt
aa|2a|ff
aa|3a|kk
aa|5a|zz
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$cat test
sept
aug
jan
oct
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$sort -M test
jan
aug
dir3 : bash

The comm command

Common - compare two sorted files line by line and write to standard output:
the lines that are common, plus the lines that are unique.

SYNOPSIS

`comm [`*`options`*`]... `*`File1 File2`*

Before `comm' can be used, the input files must be sorted using the collating sequence specified by the 'LC_COLLATE' locale, with trailing newlines significant. If an input file ends in a non-newline character, a newline is silently appended. The 'sort' command with no options always outputs a file that is suitable input to 'comm'.

With no options, `comm' produces three column output. Column one contains lines unique to FILE1, column two contains lines unique to FILE2, and column three contains lines common to both files. Columns are separated by a single TAB character.

The options -1, -2, and -3 suppress printing of the corresponding columns.

Unlike some other comparison utilities, `comm' has an exit status that does not depend on the result of the comparison. Upon normal completion `comm' produces an exit code of zero. If there is an error it exits with nonzero status.

If the file words.txt contains a subset of countries.txt then the above will return nothing.
If the file words.txt contains items that don't exist in countries.txt then the above will return those unique items.

```
$ comm -23 <(sort words.txt | uniq) <(sort countries.txt | uniq)
```

To return the lines that are in both words.txt and countries.txt use:

```
$ comm -12 <(sort words.txt | uniq) <(sort countries.txt | uniq)
```

DESCRIPTION

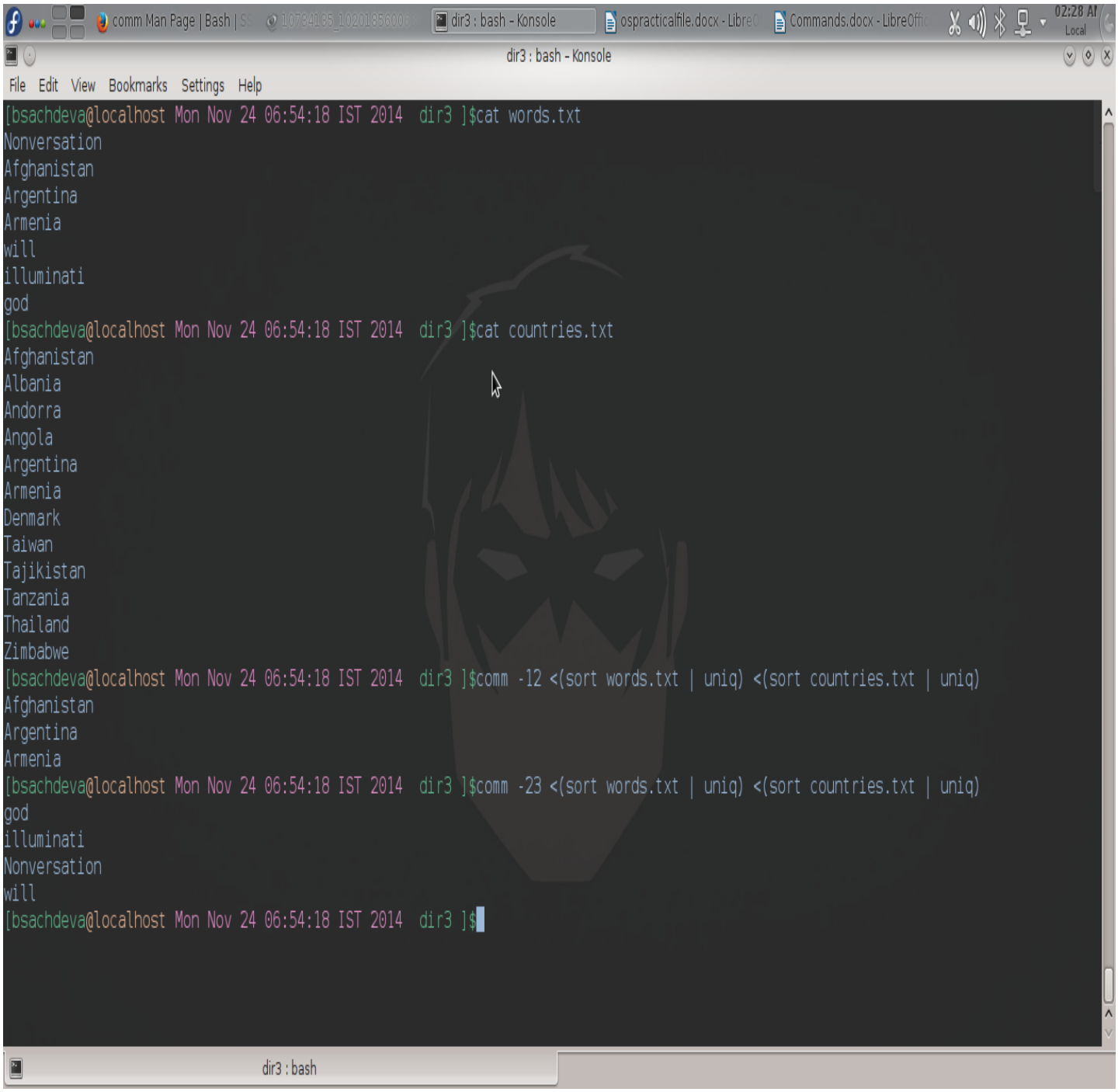
Compare sorted files FILE1 and FILE2 line by line.

With no options, produce three-column output. Column one contains lines unique to FILE1, column two contains lines unique to FILE2, and column three contains lines common to both files.

-1 suppress column 1 (lines unique to FILE1)

-2 suppress column 2 (lines unique to FILE2)

-3 suppress column 3 (lines that appear in both files)



The diff command

Display the differences between two files, or each corresponding file in two directories.
Each set of differences is called a "diff" or "patch". For files that are identical, diff normally produces no output; for binary (non-text) files, diff normally reports only that they are different.

SYNOPSIS

diff [OPTION]... FILES

In the simplest case, diff compares the contents of the two files *from-file*and *to-file*. A file name of - stands for text read from the standard input.

If *from-file* is a directory and *to-file* is not, diff compares the file in *from-file* whose file name is that of *to-file*, and vice versa. The non-directory file must not be -.

If both *from-file* and *to-file* are directories, diff compares corresponding files in both directories, in alphabetical order; this comparison is not recursive unless the -r or --recursive option is given.

GNU `diff' can show whether files are different without detailing the differences.
It also provides ways to suppress certain kinds of differences that are not important to you.

Most commonly, such differences are changes in the amount of white space between words or lines. `diff' also provides ways to suppress differences in alphabetic case or in lines that match a regular expression that you provide.

DESCRIPTION

--normal

output a normal diff (the default)

-q, --brief

report only when files differ

-s, --report-identical-files

report when two files are the same

-y, --side-by-side

output in two columns

-W, --width=NUM

output at most NUM (default 130) print columns

--left-column

output only the left column of common lines

--suppress-common-lines

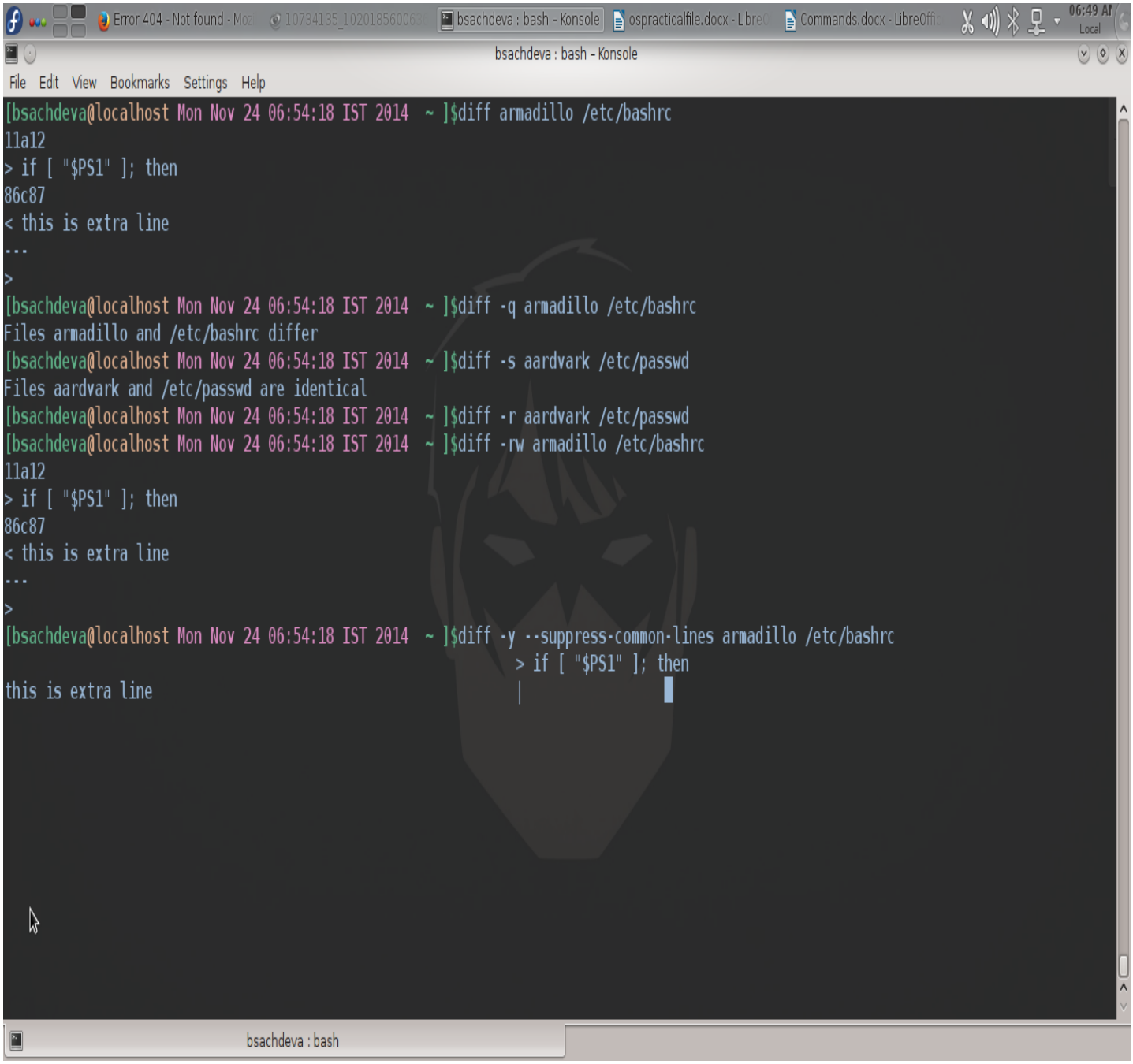
do not output common lines

-r, --recursive

recursively compare any subdirectories found

-w, --ignore-all-space

ignore all white space



The grep Command

grep is used to search text for patterns specified by the user. It is one of the most useful and powerful commands on Linux and other Unix-like operating systems.

SYNOPSIS

grep [option(s)] pattern [file(s)]

The items in square brackets are optional. When used with no options and no arguments (i.e., input files), grep searches standard input (which by default is text typed in at the keyboard) for the specified pattern and returns each line that contains a match to standard output (which by default is the display screen).

A line of text is defined in this context not as what appears as a line of text on the display screen but rather as all text between two newline characters. Newline characters are invisible characters that are represented in Unix-like operating systems by a backslash followed by the letter n and which are created when a user presses the ENTER key when using a text editor (such as gedit). Thus, a line of text returned by grep can be as short as a single character or occupy many lines on the display screen.

grep can search any number of files simultaneously. Thus, for example, the following would search the three files file1, file2 and file3 for any line that contains the string (i.e., sequence of characters) Lin:

```
grep Lin file1 file2 file3
```

Each result is displayed beginning on a separate line, and it is preceded by the name of the file in which it was found in the case of multiple files. The inclusion of the file names in the output data can be suppressed by using the -h option.

grep is not limited to searching for just single strings. It can also search for sequences of strings, including phrases. This is accomplished by enclosing the sequence of strings that forms the pattern in quotation marks (either single or double). Thus, the above example could be modified to search for the phrase Linux is:

```
grep 'Linux is' file1 file2 file3
```

Text searches with grep can be considerably broadened by combining them with wildcards and/or performing recursive searches. A wildcard is a character that can represent some specific class of characters or sequence of characters. The following is a modification of the above example that uses the star wildcard (i.e., an asterisk), which represents any character or sequence of characters, to search all text files in the current directory (i.e., the directory in which the user is currently working):

```
grep 'Linux is' *
```

grep's search area can be broadened even further by using its -r option to search recursively through an entire directory tree (i.e., a directory and all levels of subdirectories within it) rather than just the files within a specified directory. For example, the following would search all files in the current directory and in all of its subdirectories (including their subdirectories, etc.) for every line containing the full name of the creator of Linux:

```
grep -r 'Linus Torvalds' *
```

One of the most commonly employed of grep's many options is -i, which instructs it to ignore case, that is, to ignore whether letters in the pattern and text searched are lower case (i.e., small letters) or upper case (i.e., capital letters). Thus, for instance, the previous example could very easily be converted to a case-insensitive search as follows:

```
grep -ir 'Linus Torvalds' *
```

This would produce the same results as

```
grep -ir 'linUS torvAlds' *
```

Another frequently used option is -c, which causes grep to only report the number of times that the pattern has been matched for each file and to not display the actual lines. Thus, for instance, the following would show the total number of times that the string inu appears in a file named file4:

```
grep -c inu file4
```

Another useful option is -n, which causes grep to precede each line of output with the number of the line in the text file from which it was obtained. The -v option inverts the match; that is, it matches only those lines that do not contain the given pattern. The -w option tells grep to select only those lines that contain an entire word or phrase that matches the specified pattern. The -x option tells grep to select only those lines that match exactly the specified pattern.

The -l option tells grep to not return the lines containing matches but to only return only the names of the files that contain matches. The -L option is the opposite of the -l option (and analogous to the -v option) in that it will cause grep to return only the names of files that do not contain the specified pattern.

grep does not search the names of files for a specified pattern, only the text contained within files. However, sometimes it is useful to search the names of files, as well as of directories and links, rather than the contents of files. Fortunately, this can easily be accomplished by first using the ls command to list the contents of a directory and then using a pipe (which is represented by the vertical bar character) to transfer its output to grep for searching. For example, the following would provide a list of all files, directories and links in the current directory that contain the string linu in their names:

```
ls | grep linux
```

The following example uses ls with its -l (i.e., long) option (which is unrelated to grep's -l option) to find all filesystem objects in the current directory whose permissions have been set so that any user can read, write and execute them:

```
ls -l | grep rwxrwxrwx
```

grep is very useful for obtain information from log and configuration files. For example, it can be used to obtain information about the USB (universal serial bus) devices on a system by filtering the output from the dmesg command (which provides the messages from the kernel as a system is booting up) as follows:

dmesg | grep -i usb

Among grep's other options are --help, which provides a very compact summary of some of its many capabilities, and -V, or --version, which provides information about the currently installed version.

grep's search functionality can be even further refined through the use of regular expressions. These are a pattern matching system that uses strings constructed according to pre-defined syntax rules to find desired patterns in text. Additional information about grep, including its use with regular expressions, can be obtained from its built-in manual page by using the man command, i.e.,

man grep

The name grep comes from a command in ed, which was the original text editor on the UNIX operating system. The command takes the form g/re/p, which means to search globally for matches to the regular expression (i.e., re), and print (which is UNIX terminology for write on the display screen) lines that are found.

DESCRIPTION

grep searches the named input FILEs (or standard input if no files are named, or if a single hyphen-minus (-) is given as file name) for lines containing a match to the given PATTERN. By default, grep prints the matching lines. In addition, two variant programs egrep and fgrep are available. egrep is the same as grep -E. fgrep is the same as grep -F. Direct invocation as either egrep or fgrep is deprecated, but is provided to allow historical applications that rely on them to run unmodified.

-E, --extended-regexp

Interpret PATTERN as an extended regular expression (ERE, see below). (-E is specified by POSIX.)

-e PATTERN, --regexp=PATTERN

Use PATTERN as the pattern. This can be used to specify multiple search patterns, or to protect a pattern beginning with a hyphen (-)

-f FILE, --file=FILE

Obtain patterns from FILE, one per line. The empty file contains zero patterns, and therefore matches nothing. (-f is specified by POSIX.)

-i, --ignore-case

Ignore case distinctions in both the PATTERN and the input files. (-i is specified by POSIX.)

-w, --word-regexp

Select only those lines containing matches that form whole words. The test is that the

matching sub string must either be at the beginning of the line, or preceded by a non-word constituent character. Similarly, it must be either at the end of the line or followed by a non-word constituent character. Word-constituent characters are letters, digits, and the underscore.

-q, --quiet, --silent

Quiet; do not write anything to standard output. Exit immediately with zero status if any match is found, even if an error was detected. Also see the -s or --no-messages option.

-n, --line-number

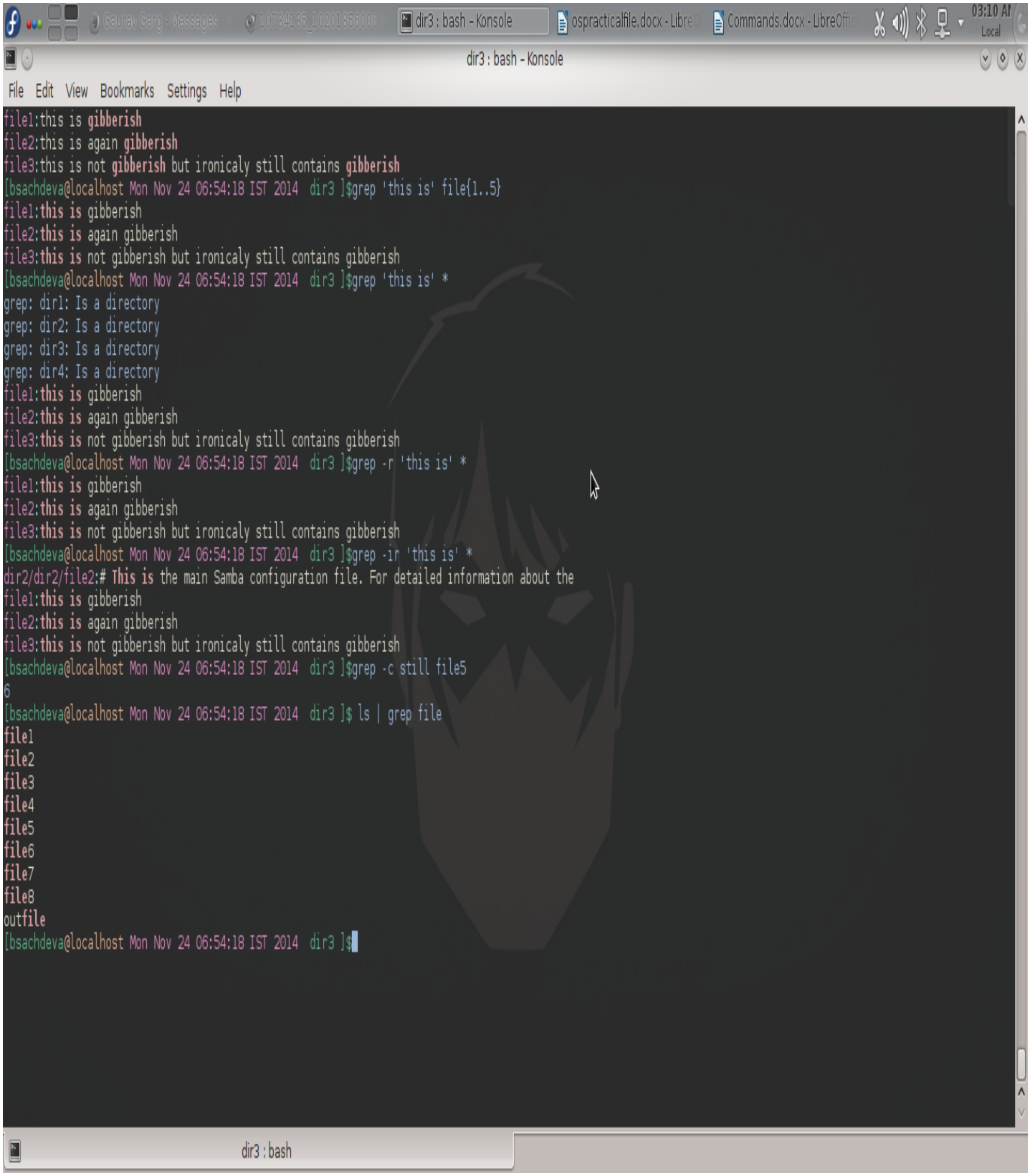
Prefix each line of output with the 1-based line number within its input file.

-a, --text

Process a binary file as if it were text; this is equivalent to the --binary-files=text option.

-c, --count

Suppress normal output; instead print a count of matching lines for each input file. With the -v, --invert-match option (see below), count non-matching lines.


dir3 : bash - Konsole
File Edit View Bookmarks Settings Help
file1:this is gibberish
file2:this is again gibberish
file3:this is not gibberish but ironicaly still contains gibberish
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$grep 'this is' file{1..5}
file1:this is gibberish
file2:this is again gibberish
file3:this is not gibberish but ironicaly still contains gibberish
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$grep 'this is' *
grep: dir1: Is a directory
grep: dir2: Is a directory
grep: dir3: Is a directory
grep: dir4: Is a directory
file1:this is gibberish
file2:this is again gibberish
file3:this is not gibberish but ironicaly still contains gibberish
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$grep -r 'this is' *
file1:this is gibberish
file2:this is again gibberish
file3:this is not gibberish but ironicaly still contains gibberish
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$grep -ir 'this is' *
dir2/dir2/file2:# This is the main Samba configuration file. For detailed information about the
file1:this is gibberish
file2:this is again gibberish
file3:this is not gibberish but ironicaly still contains gibberish
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$grep -c still file5
6
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ ls | grep file
file1
file2
file3
file4
file5
file6
file7
file8
outfile
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$
dir3 : bash

The whatis Command

The whatis command provides very brief descriptions of command line programs (i.e., all-text mode programs) and other topics related to Linux and other Unix-like operating systems.

It accomplishes this by searching the short descriptions in the whatis database for each keyword provided to it as an argument (i.e., input data). This database contains just the title, section number and description from the NAME section of each page in the man manual that is built into most Unix-like systems.

SYNOPSIS

whatis keyword(s)

For example, the following provides a single line summary of the head command (which by default displays the first ten lines of each file that is provided to it as an argument):

whatis head

whatis can be used to simultaneously search for information about multiple topics. For example, the following would provide information about both head and tail (which by default reads the final ten lines of files):

whatis head tail

The output of whatis is limited by the fact that it provides only a single line for each keyword found in the database; thus it supplies incomplete information about even moderately complex commands. For example, the following use of whatis to obtain information about the cat command generates the output "concatenate files and print on the standard output":

whatis cat

However, this omits some important information about cat, particularly the facts that it is very convenient to use for reading files and that it is also used to create and write to files.

whatis is similar to the apropos command. However, apropos is more powerful in that its arguments are not limited to complete words but can also be strings (i.e., any finite sequences of characters) which comprise parts of words. Both commands are unusual in that they have no options.

The man command (which is used to read the built-in manual pages), when used with its -f option, produces the same output as whatis. Thus, for example,

man -f cat

is equivalent to

whatis cat

The whatis database is a plain text (i.e., human-readable characters) file that is generated automatically

by the makewhatis program. Its location varies according to the particular system; in the case of Red Hat Linux 9, for example, its full path is /var/cache/man/whatis.

If desired, the database can be viewed using a command such as the following:

cat /var/cache/man/whatis | less

In this command the output of cat /var/cache/man/whatis is piped (i.e., sent) to the less command in order to make it easier to read by displaying only a single page at a time rather than scrolling down the screen at high speed.

DESCRIPTION

Each manual page has a short description available within it. %whatis% searches the manual page names and displays the manual page descriptions of any name matched name may contain wildcards (-w) or be a regular expression (-r). Using these options, it may be necessary to quote the name or escape (\) the special characters to stop the shell from interpreting them index databases are used during the search, and are updated by the %mandb% program. Depending on your installation, this may be run by a periodic cron job, or may need to be run manually after new manual pages have been installed.

OPTIONS

-v, --verbose

Print verbose warning messages.

-r, --regex

Interpret each name as a regular expression. If a name matches any part of a page name, a match will be made. This option causes %whatis% to be somewhat slower due to the nature of database searches.

-w, --wildcard

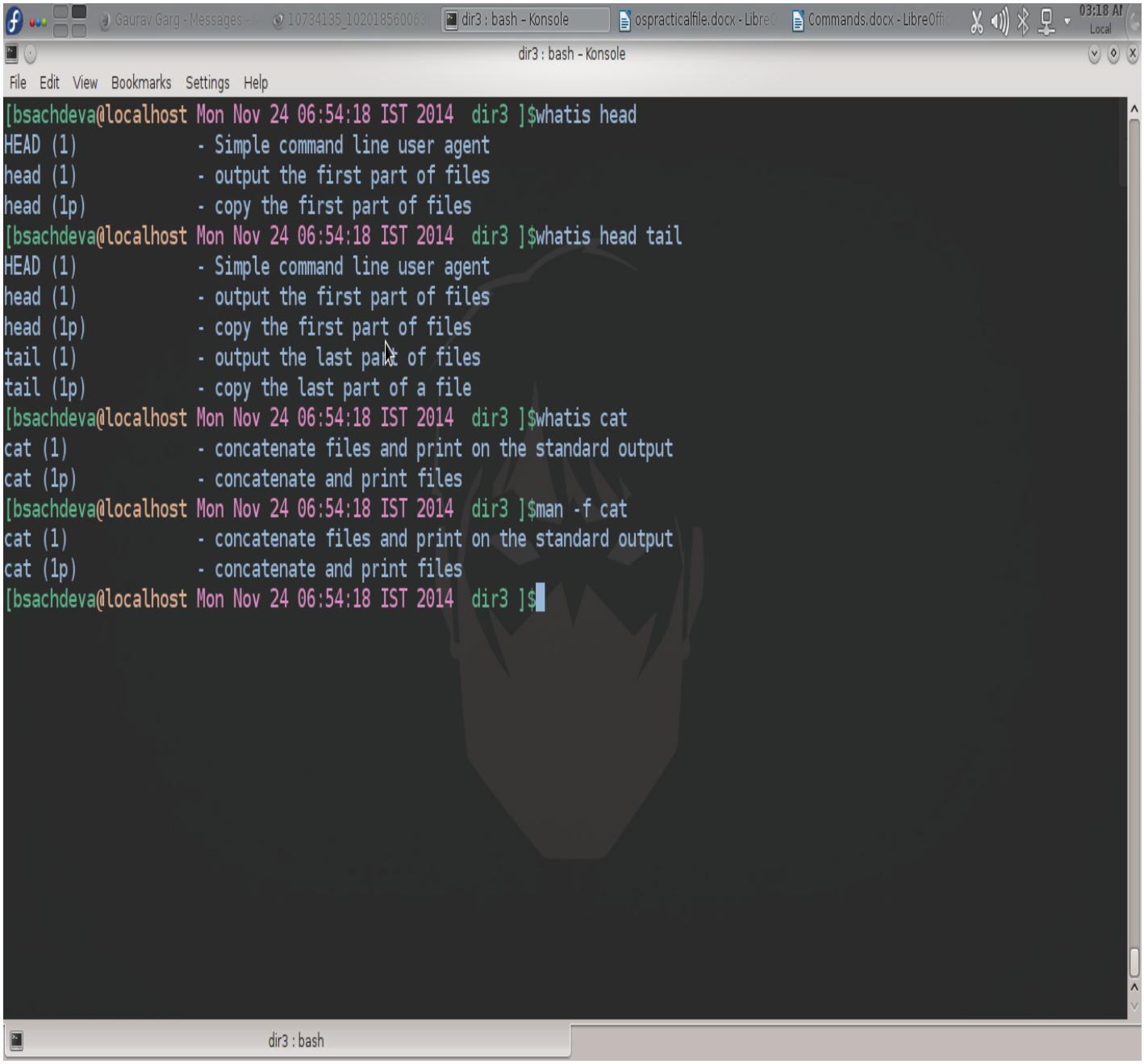
Interpret each name as a pattern containing shell style wildcards. For a match to be made, an expanded name must match the entire page name. This option causes %whatis% to be somewhat slower due to the nature of database searches.

-l, --long

Do not trim output to the terminal width. Normally, output will be truncated to the terminal width to avoid ugly results from poorly-written NAME sections.

-s list, --sections list, --section list

Search only the given manual sections. list is a colon- or comma-separated list of sections. If an entry in list is a simple section, for example "3", then the displayed list of descriptions will include pages in sections "3", "3perl", "3x", and so on; while if an entry in list has an extension.


dir3 : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$whatis head
HEAD (1) - Simple command line user agent
head (1) - output the first part of files
head (1p) - copy the first part of files
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$whatis head tail
HEAD (1) - Simple command line user agent
head (1) - output the first part of files
head (1p) - copy the first part of files
tail (1) - output the last part of files
tail (1p) - copy the last part of a file
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$whatis cat
cat (1) - concatenate files and print on the standard output
cat (1p) - concatenate and print files
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$man -f cat
cat (1) - concatenate files and print on the standard output
cat (1p) - concatenate and print files
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$
dir3 : bash

The whereis Command

The whereis command is used to locate the binary, the source code and the online manual page for any specified program. A binary is an executable (i.e., ready to run) form of a program. Source code is the original form of a program as written by a human using a programming language and before it has been converted by a compiler into a binary. The online manual pages, commonly referred to as man pages, are normally accessed using the man command.

SYNOPSIS

whereis [option(s)] program_name(s)

When used without any options, which is most commonly the case, whereis attempts to supply the absolute path names (i.e., the path to the root directory) for the binary, source code and man page for every program name (including command names, which are generally the same as program names) that is supplied to it as an argument (i.e., input). Any number of names can be accepted as arguments, and the results for each will be returned on a separate line.

Thus, for example, the following would be used to attempt to find the locations of the executable, source code and man page for the ls command (which is used to list the contents of any specified directory):

whereis ls

Although whereis ideally returns three pieces of information for each argument, in reality it often returns fewer and sometimes returns more. For example, it will not return a location for source code for a program on a computer for which the source code has not been installed (which is common when programs are installed in precompiled form during installation of the operating system). Likewise, there may be no man pages for some commands, or there may be more than one.

For some commands it might be the case that no executable is returned. Examples include alias and umask. This is presumably because no separate executables with these names exist and such commands are built directly into the shell (i.e., the program that executes commands typed in by a user and displays their results).

If an executable, source code or man page file exists on a computer but has not installed in a standard location, no result will be returned for it. This is because whereis only searches in such locations. An example is the situation in which a user has installed a program in its home directory (i.e., the directory which contains configuration files, programs and data specific to that user) rather than in directories that are accessible to all users.

In the event that nothing is found for one or more of the three types of information for which whereis searches for each argument, no error message or other notification is provided. If multiple results are found, whereis returns all of them.

In some situations, whereis might return only a single entry for some commands. This is true in the

case of the spell command (which, as its name implies, is used to check the spelling of text files), at least on some systems, as can be seen by running the following:

whereis spell

Several options are available to limit the type of results returned by whereis (as is the case with the wc command, which by default counts the number of lines, words and characters that are contained in text). The -b option tells it to search only for binaries. The -m option tells it to search only for man pages. The -s option tells it to search only for sources. Thus, for example, the following could be used to search for only the binaries and source code for the whoami command (which reports the owner of the current login session):

whereis -b whoami

As is generally (but not always) the case with single-letter options for shell programs, these options can be used together in any combination and any order. The order in which the results are displayed is not affected by the order in which the options are used.

DESCRIPTION

whereis locates the binary, source and manual files for the specified command names. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form .ext (for example: .c) Prefixes of s. resulting from use of source code control are also dealt with. whereis then attempts to locate the desired program in the standard Linux places, and in the places specified by $PATH and $MANPATH.

-b Search only for binaries.

-m Search only for manuals.

-s Search only for sources.

-u Only show the command names that have unusual entries. A command is said to be unusual if it does not have just one entry of each explicitly requested type. Thus 'whereis -m -u *' asks for those files in the current directory which have no documentation file, or more than one.

-B list

Limit the places where whereis searches for binaries, by a whitespace-separated list of directories.

-M list

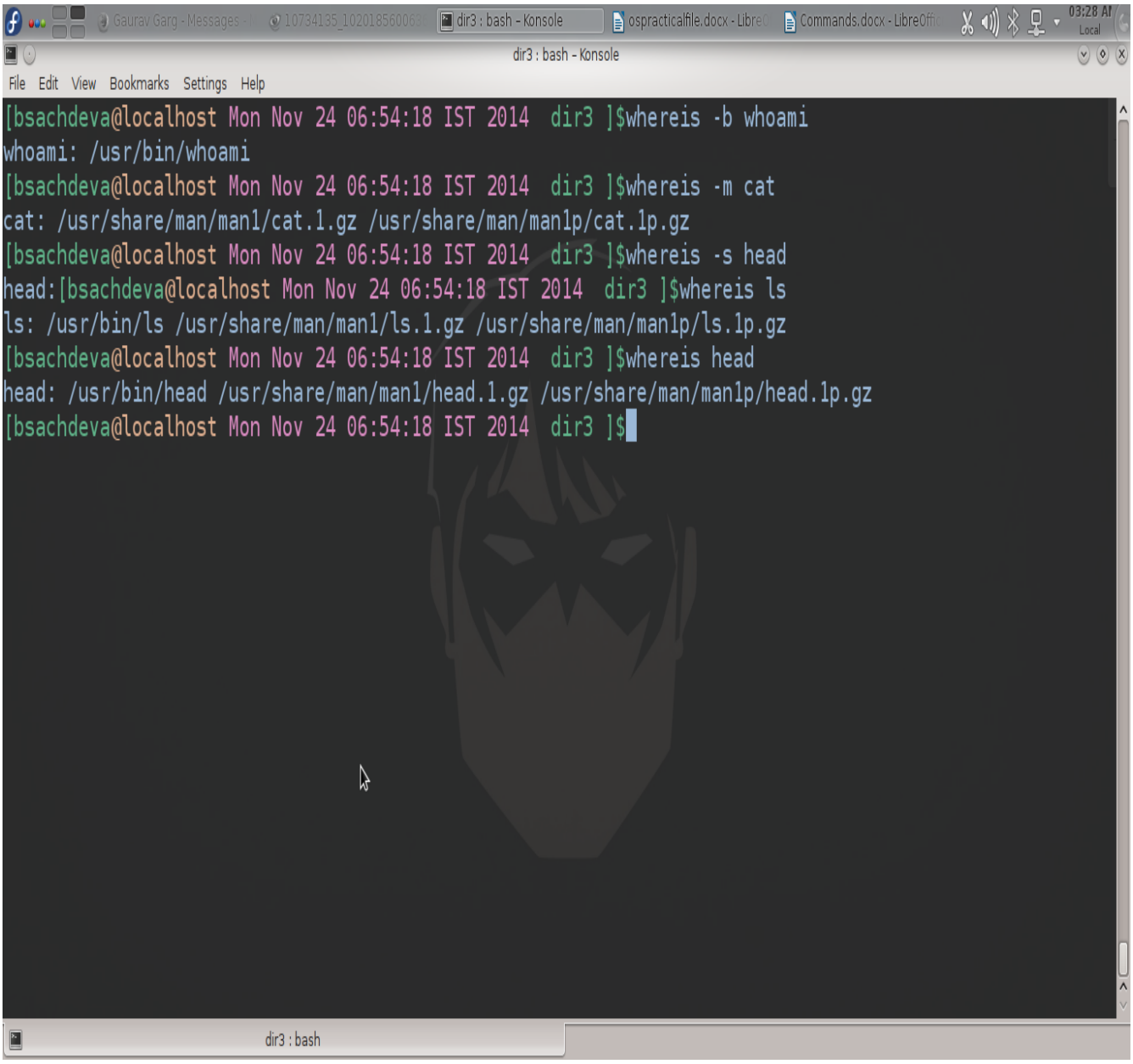
Limit the places where whereis searches for manuals, by a whitespace-separated list of directories.

-S list

Limit the places where whereis searches for sources, by a whitespace-separated list of directories.

-f Terminates the directory list and signals the start of filenames. It must be used when any of the -B, -M, or -S options is used.

-l Output list of effective lookup paths the whereis is using. When non of -B, -M, or -S is specified the option will out hard coded paths that the command was able to



find on system.

The bzip2 Command

The bzip2 command is used for compressing and decompressing files.

Data compression, also referred to as just compression, is the process of encoding data using fewer bits. Data decompression, or just decompression, is the process of restoring compressed data back into a form in which it is again useful. bzip2 features a high rate of compression together with reasonably fast speed. Most files can be compressed to a smaller file size than is possible with the more traditional gzip and zip programs. Moreover, like those programs, the compression is lossless, meaning that no data is lost during compression and thus the original files can be exactly regenerated. The only disadvantage of bzip2 is that it is somewhat slower than gzip and zip. This performance has been made possible through the use of the Burrows-Wheeler transform (BWT). First published by Michael Burrows and David Wheeler in 1994, BWT is not by itself a data compression algorithm; rather, it converts blocks of data into a format that is extremely well suited for compression. In addition, it is reversible, which means that the original data can be easily regenerated.

SYNOPSIS

bzip2 [option(s)] file_name(s)

bzip2 is commonly used without any options. Any number of files can be compressed simultaneously by merely listing their names as arguments (i.e., inputs). Thus, for example, the following would compress the three files named file1, file2 and file3:

bzip2 file1 file2 file3

If no problems are encountered, by default no confirmation is provided. However, if some problem is encountered, an error message is returned for each file for which there is a problem. Should confirmation and data on the degree of compression be desired, then bzip2 can be used with its -v (i.e., verbose) option, such as

bzip2 -v file1 file2 file3

Each file is replaced by a compressed version of itself, with the extension .bz2 appended to its name. Thus, in the case of the above example, the three input files would be replaced by files named file1.bz2, file2.bz2 and file3.bz2. This can be easily confirmed with the ls (i.e., list) command, and the sizes of the new files can be viewed by using ls together with its -s (i.e., size) option. The original files can be retained by using the -k (i.e., keep) option.

Each compressed file retains to the extent possible the same metadata (i.e., modification date, access permissions and ownership) as the corresponding original file, so that this data can be correctly restored at decompression time.

Compression is performed even if the compressed file is larger than the original. This situation can

occur in the case of very small files, because the compression mechanism has a fixed overhead of about 50 bytes.

The -s (i.e., small) option reduces memory usage for compression, decompression and testing, which is useful for computers with very small memories (e.g., 8MB or less). Files are decompressed and tested using a modified algorithm that only requires 2.5 bytes per block, although this results in a reduction in speed by about half.

The -t (i.e., test) option tells bzip2 to check the integrity of the specified file(s) by performing a trial decompression and discarding the result. There is no effect on the specified files.

As a safety measure, by default bzip2 will not overwrite existing files with the same names (inclusive of extensions) as its output files or break hard links (i.e., alternative names for a single file). However, the -f option forces the overwriting of existing files and also instructs bzip2 to break hard links to files.

The -d option tells bzip2 to decompress the specified files. It is the same as the bunzip2 command. Files that were not created by bzip2 will be detected and ignored, and a warning will be issued. An attempt is made to guess the original name for each file being decompressed; however, if the compressed version does not end in one of the recognized extensions (i.e., .bz2, .bz, .tbz2 or .tbz), then bzip2 will complain that it cannot guess the name of the original file and will use the name of the compressed version with the .out extension appended to it.

It is often convenient to store a number of files in a compressed archive, which is a single file that contains any number of individual files plus information to allow them to be restored to their original form by one or more extraction programs. This can be accomplished by first combining the uncompressed files using the tar command together with its -c and -f options and then using bzip2 on the combined file. The -c option instructs tar to combine the files, and the -f option tells it to use the argument that follows as the name of the new file. For example, the following commands would combine three files named file4, file5 and file6 into a single archive named file7.tar and then compress that archive into a compressed file named file7.tar.bz2:

```
tar -cf file7.tar file4 file5 file6

bzip2 file7.tar
```

Alternatively, this can all be accomplished in a single step by merely adding the -j option to tar, which tells tar to compress the archive that it creates with bzip2. Thus, the above example would be simplified to:

```
tar -cjf file7.tar.bz2 file4 file5 file6
```

DESCRIPTION

bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by more conventional LZ77/LZ78-based compressors, and approaches the performance of the PPM family of statistical compressors.

OPTIONS

-c --stdout

Compress or decompress to standard output.

-d --decompress

Force decompression. bzip2, bunzip2 and bzcat are really the same program, and the decision about what actions to

take is done on the basis of which name is used. This flag overrides that mechanism, and forces bzip2 to decompress.

-z --compress

The complement to -d: forces compression, regardless of the invocation name.

-t --test

Check integrity of the specified file(s), but don't decompress them. This really performs a trial decompression and throws away the result.

-f --force

Force overwrite of output files. Normally, bzip2 will not overwrite existing output files. Also forces bzip2 to break hard links to files, which it otherwise wouldn't do.

bzip2 normally declines to decompress files which don't have the correct magic header bytes. If forced (-f), however, it will pass such files through unmodified. This is how GNU gzip behaves.

-q --quiet

Suppress non-essential warning messages. Messages pertaining to I/O errors and other critical events will not be

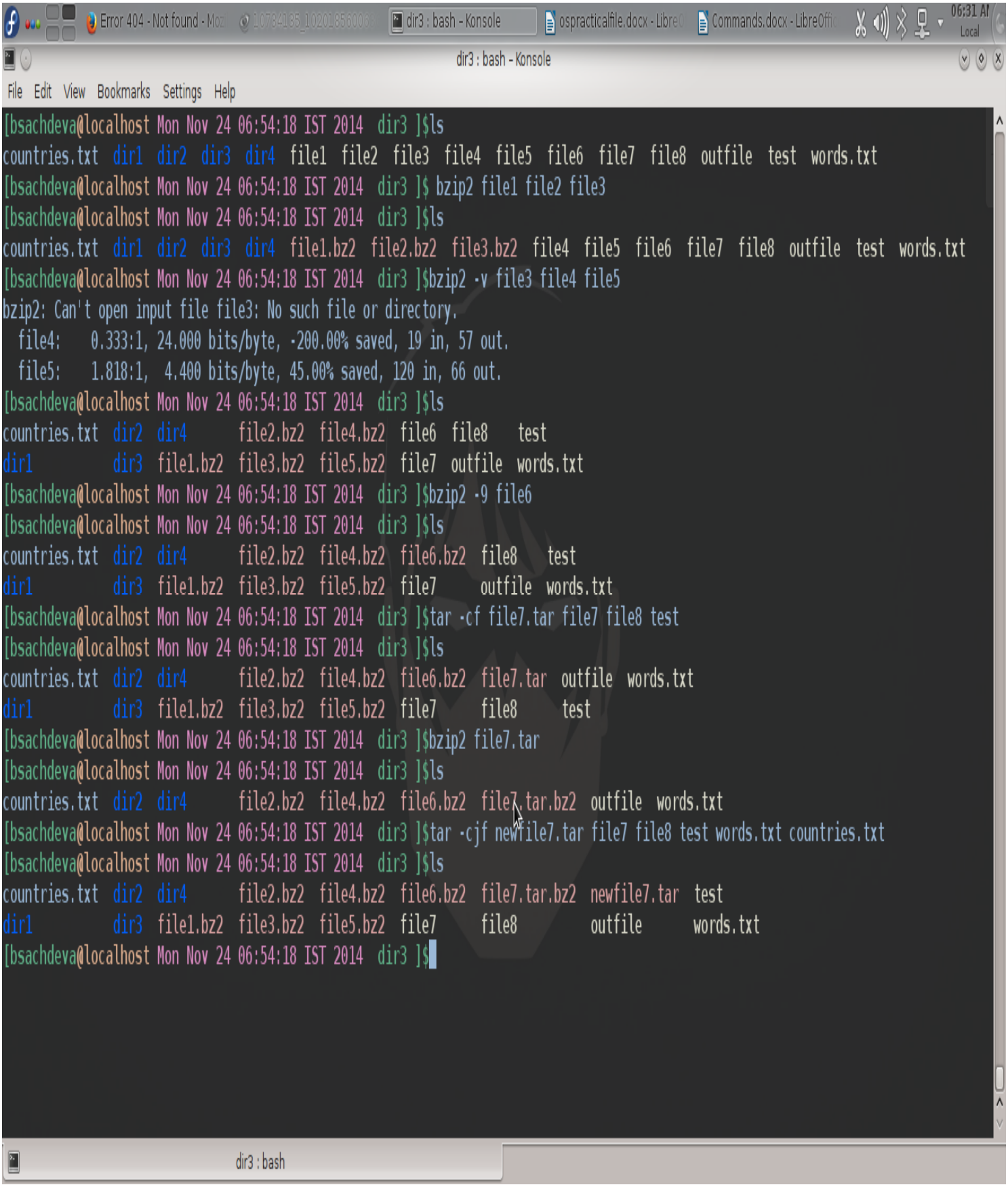
suppressed.

-v --verbose

Verbose mode -- show the compression ratio for each file processed. Further -v's increase the verbosity level, spewing out lots of information which is primarily of interest for diagnostic purposes.

-1 (or --fast) to -9 (or --best)

Set the block size to 100 k, 200 k .. 900 k when compressing. Has no effect when decompressing. See MEMORY MANAGEMENT below. The --fast and --best aliases are primarily for GNU gzip compatibility. In particular, --fast doesn't make things significantly faster.


Error 404 - Not found - Mo
dir3 : bash - Konsole
ospracticalfile.docx - Libre
Commands.docx - LibreOffi
06:31 AM
Local
dir3 : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt dir1 dir2 dir3 dir4 file1 file2 file3 file4 file5 file6 file7 file8 outfile test words.txt
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ bzip2 file1 file2 file3
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt dir1 dir2 dir3 dir4 file1.bz2 file2.bz2 file3.bz2 file4 file5 file6 file7 file8 outfile test words.txt
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$bzip2 -v file3 file4 file5
bzip2: Can't open input file file3: No such file or directory.
file4: 0.333:1, 24.000 bits/byte, -200.00% saved, 19 in, 57 out.
file5: 1.818:1, 4.400 bits/byte, 45.00% saved, 120 in, 66 out.
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt dir2 dir4 file2.bz2 file4.bz2 file6 file8 test
dir1 dir3 file1.bz2 file3.bz2 file5.bz2 file7 outfile words.txt
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$bzip2 -9 file6
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt dir2 dir4 file2.bz2 file4.bz2 file6.bz2 file8 test
dir1 dir3 file1.bz2 file3.bz2 file5.bz2 file7 outfile words.txt
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$tar -cf file7.tar file7 file8 test
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt dir2 dir4 file2.bz2 file4.bz2 file6.bz2 file7.tar outfile words.txt
dir1 dir3 file1.bz2 file3.bz2 file5.bz2 file7 file8 test
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$bzip2 file7.tar
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt dir2 dir4 file2.bz2 file4.bz2 file6.bz2 file7.tar.bz2 outfile words.txt
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$tar -cjf newfile7.tar file7 file8 test words.txt countries.txt
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt dir2 dir4 file2.bz2 file4.bz2 file6.bz2 file7.tar.bz2 newfile7.tar test
dir1 dir3 file1.bz2 file3.bz2 file5.bz2 file7 file8 outfile words.txt
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$
dir3 : bash

The tar Command

The tar (i.e., tape archive) command is used to convert a group of files into an archive An archive is a single file that contains any number of individual files plus information to allow them to be restored to their original form by one or more extraction programs. Archives are convenient for storing files as well as for for transmitting data and distributing programs. Moreover, they are very easy to work with, often much more so than dealing with large numbers of individual files. Although tar was originally designed for backups on magnetic tape, it can now be used to create archive files anywhere on a filesystem. Archives that have been created with tar are commonly referred to as tarballs. Unlike some other archiving programs, and consistent with the Unix philosophy that each individual program should be designed to do only one thing but do it well, tar does not perform compression. However, it is very easy to compress archives created with tar by using specialized compression utilities.

SYNOPSIS

tar option(s) archive_name file_name(s)

tar has numerous options, many of which are not frequently used. Unlike many commands, tar requires the use of at least one option, and usually two or more are necessary.

tar files are created by using both the -c and -f options. The former instructs tar to create an archive and the latter indicates that the next argument (i.e., piece of input data in a command) will be the name of the new archive file. Thus, for example, the following would create an archive file called file.tar from the three files named file1, file2 and file3 that are located in the current directory (i.e., the directory in which the user is currently working):

tar -cf file.tar file1 file2 file3

It it not absolutely necessary that the new file have the .tar extension; however, the use of this extension can be is very convenient because it allows the type of file to be visually identified. It is necessary, however, that the -f option be the final option in a sequence of contiguous, single-letter options; otherwise, the system will become confused as to the desired name for the new file and will use the next option in the sequence as the name.

The -v (i.e., verbose) option is commonly used together with the -c and -f options in order to display a list of the files that are included in the archive. In such case, the above example would become

tar -cvf file.tar file1 file2 file3

tar can also be used to make archives from the contents of one or more directories. The result is recursive; that is, it includes all objects (e.g., directories and files) within each level of directories. For example, the contents of two directories named dir1 and dir2 could be archived into a file named dir.tar with the following:

tar -cvf dir.tar dir1 dir2

It is often convenient to use tar with a wildcard (i.e., a character which can represent some specific class of characters or sequence of characters). The following example uses the star wildcard (i.e., an asterisk), which represents any character or sequence of characters, to create an archive of every object in the current directory:

```
tar -cf *
```

By default, tar creates an archive of copies of the original files and/or directories, and the originals are retained. However, they can be removed when using tar by adding the --remove-files option.

As it has no compression and decompression capabilities of its own, tar is commonly used in combination with an external compression utility. A very handy feature of the GNU version (which is standard on Linux) is the availability of options that will cause standard compression programs to compress a new archive file as soon as it has been created. They are -j (for bzip2), -z (for gzip) and -Z (for compress). Thus, for example, the following would create an archive named files.tar.bz2 of the files file4, file5 and file6 that is compressed using bzip2:

```
tar -cvjf files.tar.bz2 file4 file5 file6
```

tar can also be used for unpacking tar files. However, before doing this, there are several steps that should be taken. One is to confirm that sufficient space is available on the hard disk drive (HDD). Another is to move to an empty directory (which usually involves creating one with an appropriate name) to prevent the reconstituted files from cluttering up the current directory and overwriting any files or directories with same names that are in it. In addition, if the archive has been compressed, it must first be decompressed using the appropriate decompression program (which can usually be determined by the filename extension).

In order to unpack a tar file, the -x (for extract) and -f options are required. It is also common to add the -v option to provide a running listing of the files being unpacked. Thus, for example, to unpack the archive file.tar created in a previous example the following would be used:

```
tar -xvf file.tar
```

Just as options are available to allow three compression programs to automatically compress newly created tar files, the same options can be used to have the compression programs automatically decompress tar files prior to extraction. Thus, for instance, the following would decompress and extract the contents of the compressed archive files.tar.bz2 that was created in an above example:

```
tar -xjvf files.tar.bz2
```

Files can be added to an existing archive using the -r option. As is always the case with tar, it is also necessary to use the -f option to indicate that the following string (i.e., sequence of characters) is the name of the archive. For example, the following would append a file named file7 to file.tar:

```
tar -rf file.tar file7
```

The --delete option allows specified files to be completely removed from a tar file (except when the tar file is on magnetic tape). However, this is different from an extraction, as copies of the removed files

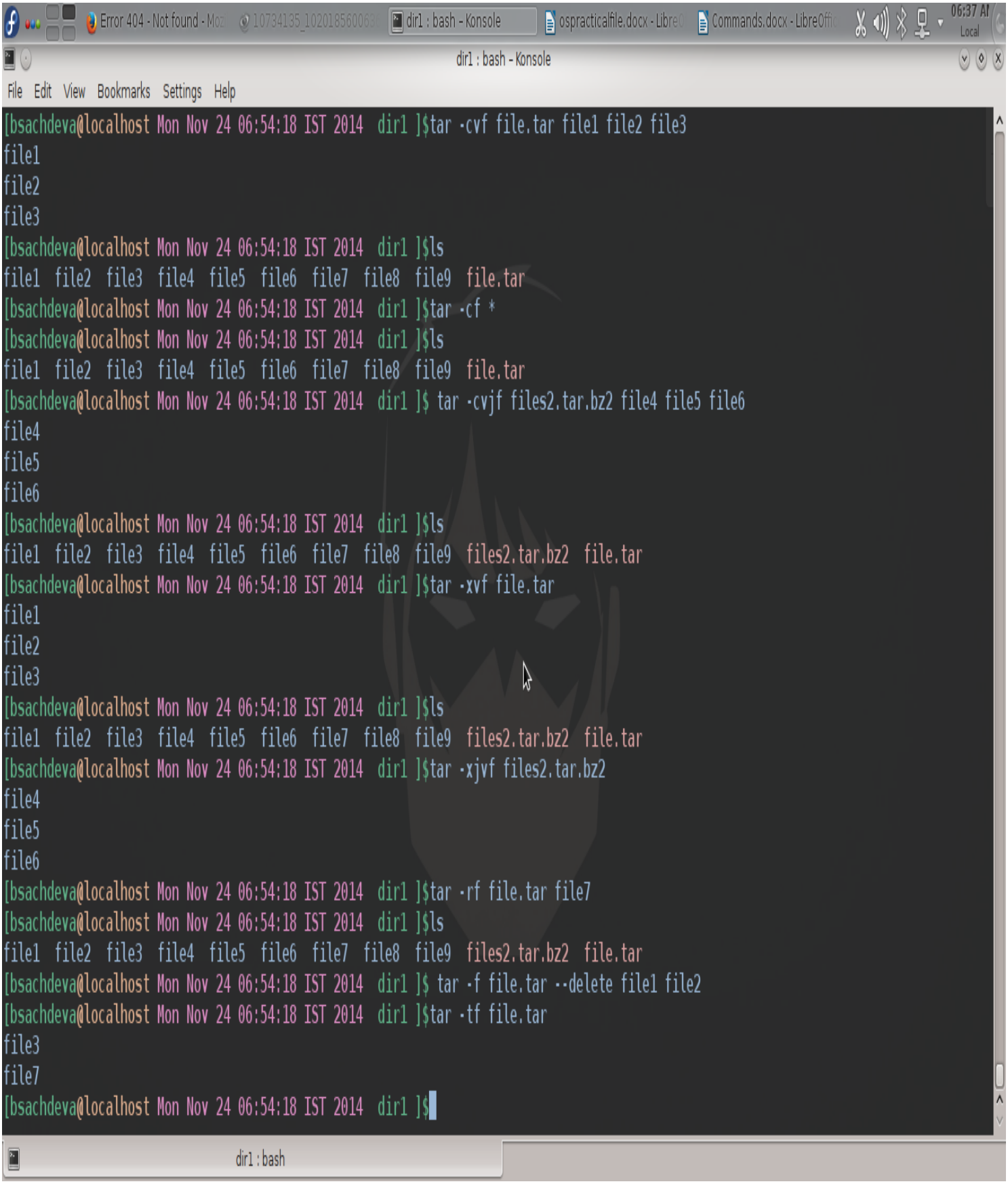
are not made and placed in the current directory. Thus, for example, the files file1 and file2 can be removed from file.tar with the following:

```
tar -f file.tar --delete file1 file2
```

The -t option tells tar to list the contents of an uncompressed archive without performing an extraction. Thus, the following would list the contents of file.tar:

```
tar -tf file.tar
```

One of the very few options that can be used alone with tar is --help, which provides a relatively compact listing of the numerous options that are available. Another is --version, which shows the version number for the installed tar program as well as its copyright information.


Error 404 - Not found - Mo
10734135_10201856006
dir1 : bash - Konsole
ospracticalfile.docx - Libre
Commands.docx - LibreOffi
06:37 AM
Local
dir1 : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$tar -cvf file.tar file1 file2 file3
file1
file2
file3
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$ls
file1 file2 file3 file4 file5 file6 file7 file8 file9 file.tar
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$tar -cf *
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$ls
file1 file2 file3 file4 file5 file6 file7 file8 file9 file.tar
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$ tar -cvjf files2.tar.bz2 file4 file5 file6
file4
file5
file6
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$ls
file1 file2 file3 file4 file5 file6 file7 file8 file9 files2.tar.bz2 file.tar
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$tar -xvf file.tar
file1
file2
file3
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$ls
file1 file2 file3 file4 file5 file6 file7 file8 file9 files2.tar.bz2 file.tar
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$tar -xjvf files2.tar.bz2
file4
file5
file6
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$tar -rf file.tar file7
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$ls
file1 file2 file3 file4 file5 file6 file7 file8 file9 files2.tar.bz2 file.tar
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$ tar -f file.tar --delete file1 file2
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$tar -tf file.tar
file3
file7
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir1]$
dir1 : bash

The rm Command

The rm (i.e., remove) command is used to delete files and directories on Linux and other Unix-like operating systems.

SYNOPSIS

rm [options] [-r directories] filenames

The items in square brackets are optional. When used just with the names of one or more files, rm deletes all those files without requiring confirmation by the user. Thus, in the following example, rm would immediately delete the files named file1, file2 and file3, assuming that all three are located in the current directory (i.e., the directory in which the user is currently working):

rm file1 file2 file3

Error messages are returned if a file does not exist or if the user does not have the appropriate permission to delete it. Write-protected files prompt the user for a confirmation (with a y for yes and an n for no) before removal. Files located in write-protected directories can never be removed, even if those files are not write-protected.

The -f (i.e., force) option tells rm to remove all specified files, whether write-protected or not, without prompting the user. It does not display an error message or return error status if a specified file does not exist. However, if an attempt is made to remove files in a write-protected directory, this option will not suppress an error message.

The -i (i.e., interactive) option tells rm to prompt the user for confirmation before removing each file and directory. If both the -f and -i options are specified, the last one specified takes affect.

As a safety measure, rm does not delete directories by default. In order to delete directories, it is necessary to use the -r option, which is the same as the -R option. This option recursively removes directories and their contents in the argument list; that is, the specified directories will first be emptied of any subdirectories (including their subdirectories and files, etc.) and files and then removed. The user is normally prompted for removal of any write-protected files in the directories unless the -f option is used.

If a file encountered by rm is a symbolic link, the link is removed, but the file or directory to which that link refers will not be affected. A user does not need write permission to delete a symbolic link, as long as the user has write permission for the directory in which that link resides.

The rm command supports the -- (two consecutive dashes) parameter as a delimiter that indicates the end of the options. This is useful when the name of a file or directory begins with a dash or hyphen. For example, the following removes a directory named -dir1:

rm -r -- -dir1

Other options include -v (i.e., verbose), which provides additional information about what is happening, --help, which provides basic documentation about rm, and --version, which tells the version of rm that is currently in use. Some differences exist among the various versions of rm, so it is always wise to read the documentation for the particular system.

The rmdir command differs from rm in that it is only used to remove empty directories.

DESCRIPTION

-f, --force

ignore nonexistent files and arguments, never prompt

-i prompt before every removal

-I prompt once before removing more than three files, or when removing recursively. Less intrusive than -i, while still giving protection against most mistakes

--interactive[=WHEN]

prompt according to WHEN: never, once (-I), or always (-i). Without WHEN, prompt always

--no-preserve-root

do not treat '/' specially

--preserve-root

do not remove '/' (default)

-r, -R, --recursive

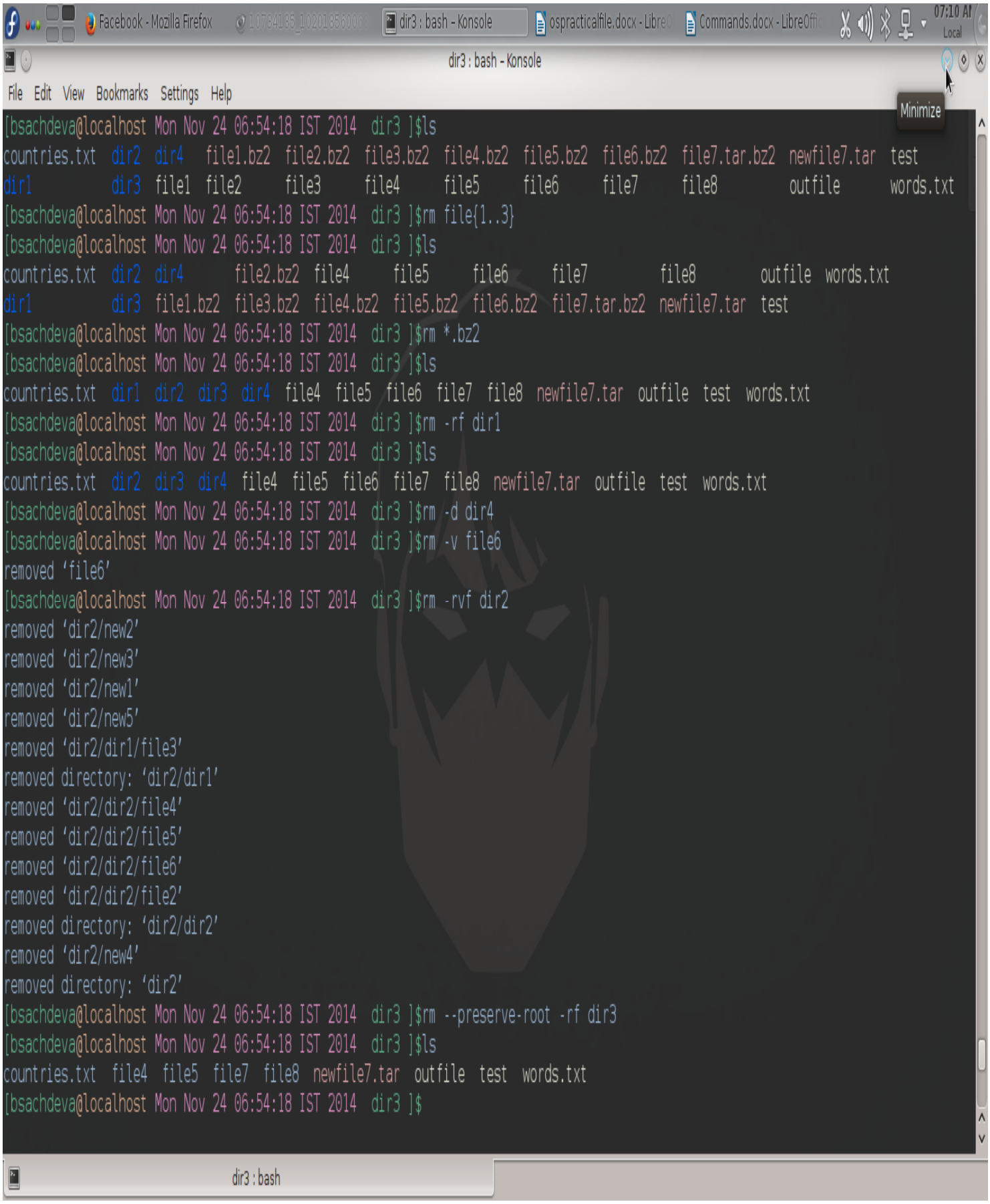
remove directories and their contents recursively

-d, --dir

remove empty directories

-v, --verbose

explain what is being done


Facebook - Mozilla Firefox
dir3 : bash - Konsole
ospracticalfile.docx - Libre
Commands.docx - LibreOffi
07:10 AM
Local
dir3 : bash - Konsole
File Edit View Bookmarks Settings Help
Minimize
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt dir2 dir4 file1.bz2 file2.bz2 file3.bz2 file4.bz2 file5.bz2 file6.bz2 file7.tar.bz2 newfile7.tar test
dir1 dir3 file1 file2 file3 file4 file5 file6 file7 file8 outfile words.txt
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$rm file{1..3}
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt dir2 dir4 file2.bz2 file4 file5 file6 file7 file8 outfile words.txt
dir1 dir3 file1.bz2 file3.bz2 file4.bz2 file5.bz2 file6.bz2 file7.tar.bz2 newfile7.tar test
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$rm *.bz2
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt dir1 dir2 dir3 dir4 file4 file5 file6 file7 file8 newfile7.tar outfile test words.txt
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$rm -rf dir1
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt dir2 dir3 dir4 file4 file5 file6 file7 file8 newfile7.tar outfile test words.txt
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$rm -d dir4
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$rm -v file6
removed 'file6'
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$rm -rvf dir2
removed 'dir2/new2'
removed 'dir2/new3'
removed 'dir2/new1'
removed 'dir2/new5'
removed 'dir2/dir1/file3'
removed directory: 'dir2/dir1'
removed 'dir2/dir2/file4'
removed 'dir2/dir2/file5'
removed 'dir2/dir2/file6'
removed 'dir2/dir2/file2'
removed directory: 'dir2/dir2'
removed 'dir2/new4'
removed directory: 'dir2'
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$rm --preserve-root -rf dir3
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$ls
countries.txt file4 file5 file7 file8 newfile7.tar outfile test words.txt
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 dir3]$
dir3 : bash

The tr Command

The tr command is used to translate specified characters into other characters or to delete them. In contrast to many command line programs, tr does not accept file names as arguments (i.e., input data). Instead, it only accepts inputs via standard input, (i.e., from the keyboard) or from the output of other programs via redirection.

SYNOPSIS

tr [options] set1 [set2]

The items in the square brackets are optional. tr requires at least one argument and accepts a maximum of two. The first, designated set1, lists the characters in the text to be replaced or removed. The second, set2, lists the characters that are to be substituted for the characters listed in the first argument.

When used without any options, tr will replace the characters provided in set1 with those provided in set2. Thus, for example, every instance of the letter a in text typed in at the keyboard can be replaced with the letter b by entering the following command, pressing the ENTER key to start a new line, typing the text, and then pressing the ENTER key again:

tr a b

This substitution can repeated for additional text by typing it in and then pressing the ENTER key again. It can be terminated by simultaneously pressing the CONTROL and c keys.

Although tr cannot accept the names of files as arguments, it can nevertheless be used to modify copies of their contents. All that is necessary is to use the input redirection operator, which is represented by a leftward pointing angular bracket, before the name of the file whose text is to be modified. For example, the following would redirect a copy of a file named file1 to tr, which would display its contents on the monitor screen with every instance of an a replaced with a b:

tr a b < file1

It is typically more convenient to have the modified output written to a file rather than being displayed on the screen, and this can be easily accomplished by using the output redirection operator, which is represented by a rightward pointing angular bracket. Thus, for example, the following would replace the letter c with the letter d in the text from file1 and write it to a file named file2:

tr c d < file1 > file2

If file2 does not already exist, it will be created automatically by the output redirection operator. If it already exists, it will be overwritten.

Care should be taken to avoid attempting to use the output redirection operator to write to the same file from which the text is being read, as this will erase all of the text in that file, including that outputted by tr. Thus, for example, the following should not be used:

tr c d < file1 > file1

What can be done, however, is to use the append redirection operator, which is represented by two successive rightward pointing angular brackets, to add the modified text to the end of the existing text in the input file. Thus, the above example could be modified to append the revised text to file1 as follows:

```
tr c d < file1 >> file1
```

An alternative way to obtain a copy of the content of a file is to use a command such as cat to read it and then send its output to tr via a pipe, which is represented by the vertical bar character. For example, the following sends a copy of the text in a file named file3 to tr for translation before writing the result to another file named file4:

```
cat file3 | tr c d > file4
```

tr can do much more than just translate a single letter. It can also translate any number of specified characters into other characters. In such case, each of the two sets needs to be enclosed in square brackets, and these brackets, in turn, need to be enclosed in quotation marks. Either single of double quotation marks can be used.

One way of performing such translation of multiple characters is by listing the characters that are to be changed in the first argument and and listing their respective replacements in order in the second argument. For example, the following would replace all instances of the letters e, f and g in a file named file5 with the letters x, y and z, respectively, and write the output to another file named file6:

```
cat file5 | tr '[efg]' '[xyz]' > file6
```

tr can also replace characters in a specified range by their counterparts in another specified range, which can be more convenient when numerous characters are involved. A range is indicated by inserting a hyphen between the first and last characters, placing all of this in square brackets and then enclosing the brackets in quotation marks. Thus, the above example could be rewritten as follows:

```
cat file5 | tr '[e-g]' '[x-z]' > file6
```

Likewise, the following would replace every upper case letter in a file named file7 by its lower case counterpart and write the result to a file called file8:

```
cat file7 | tr '[A-Z]' '[a-z]' > file8
```

tr can also be used with control characters. Also referred to as non-printing characters, these are bit sequences that represent basic formatting instructions, etc., and each begins with a backslash. They are \b for backspace, \f for form feed, \n for new line, \r for return, \t for horizontal tab, \v for vertical tab, \a for buzzer sound, \\ for backslash and \NNN for octal value NNN (consisting of one to three digits).

tr also has a number of standardized arguments that are often more convenient to use instead of making lists or ranges of characters to be translated. Each consists of a word or abbreviation surrounded by colons and then enclosed in a set of square brackets. For example, all of the letters of the alphabet are represented by [:alpha:], all numerals are represented by [:digit:], all alphanumeric characters are represented by [:alnum:], all horizontal white spaces are represented by [:blank:], all horizontal or

vertical white spaces are represented by [:space:], all printable characters exclusive of spaces are represented by [:graph:], all control characters are represented by [:cntrl:], all lower case letters are represented by [:lower:], all upper case letters are represented by [:upper:], all punctuation characters are represented by [:punct:], all printable characters including spaces are represented by [:print:], and all hexadecimal digits are represented by [:xdigit:].

Thus, for example, the previous example could be rewritten as

```
cat file7 | tr [:upper:] [:lower:] > file8
```

Probably the most useful of tr's several options is -s, which replaces each instance of a sequence of repeated characters listed in set1 with a single instance of the character specified in set2. This squeeze option is commonly used to replace each sequence of multiple blank spaces in text with a single blank space. For example, the following will remove all instances of successive blank spaces from a copy of the text in a file named file9 and write its output to a new file named file10:

```
tr -s ' ' ' ' < file9 > file10
```

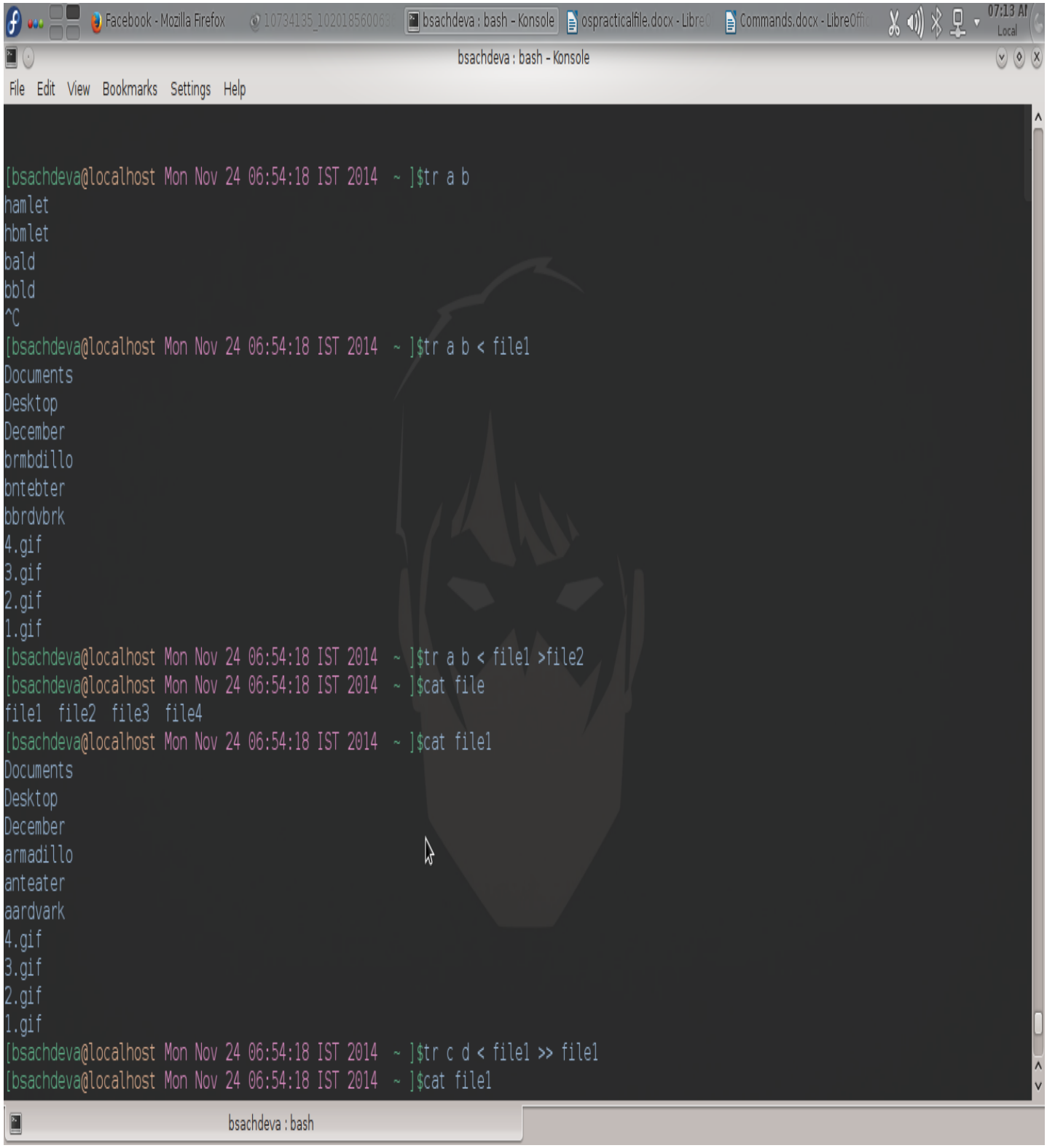
The -d option is used to delete every instance of the string (i.e., sequence of characters) specified in set1. Thus, for example, the following would remove every instance of the word soft from a copy of the text in a file named file11 and write the modified text to a file named file12:

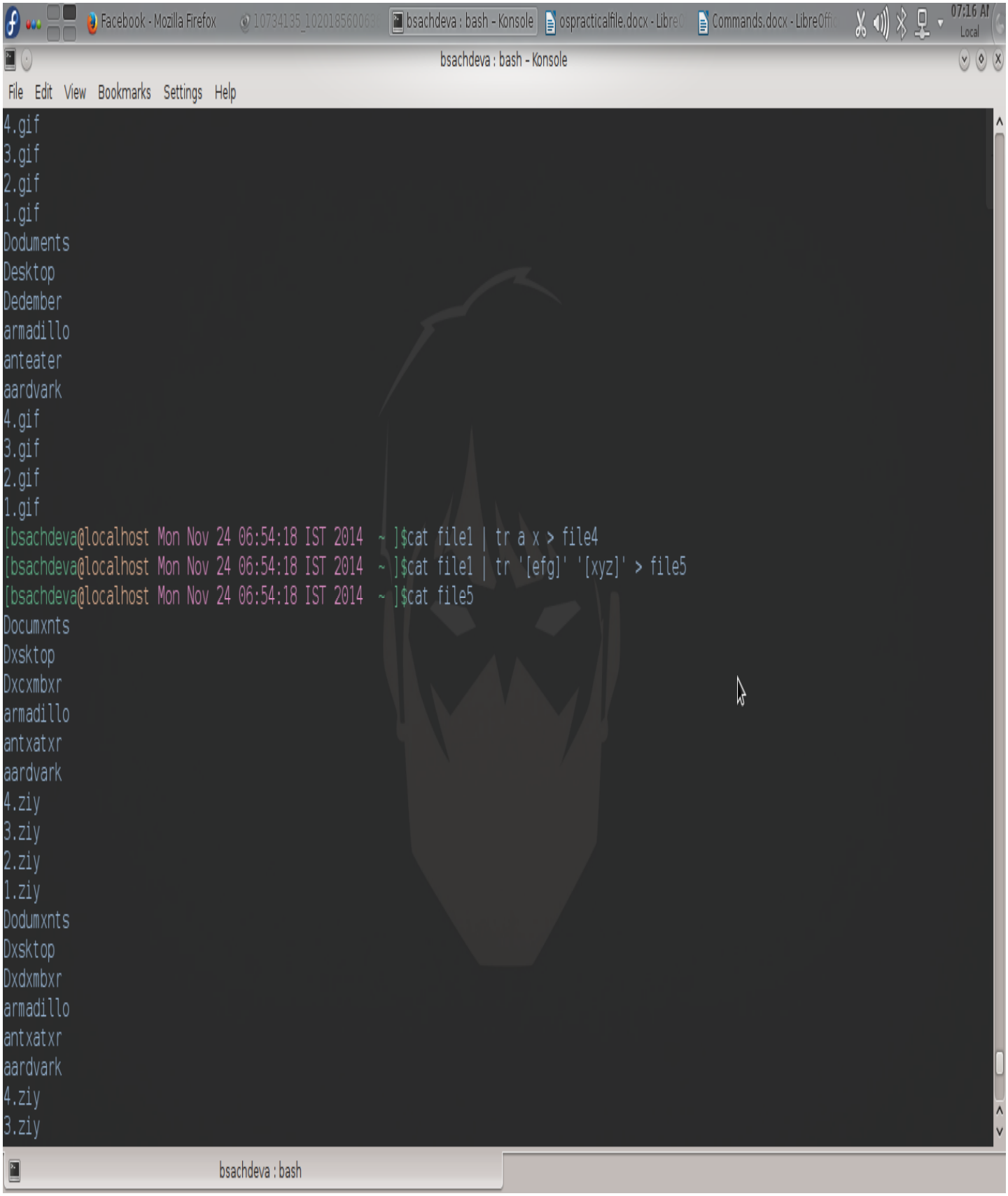
```
cat file11 | tr -d 'soft' > file12
```

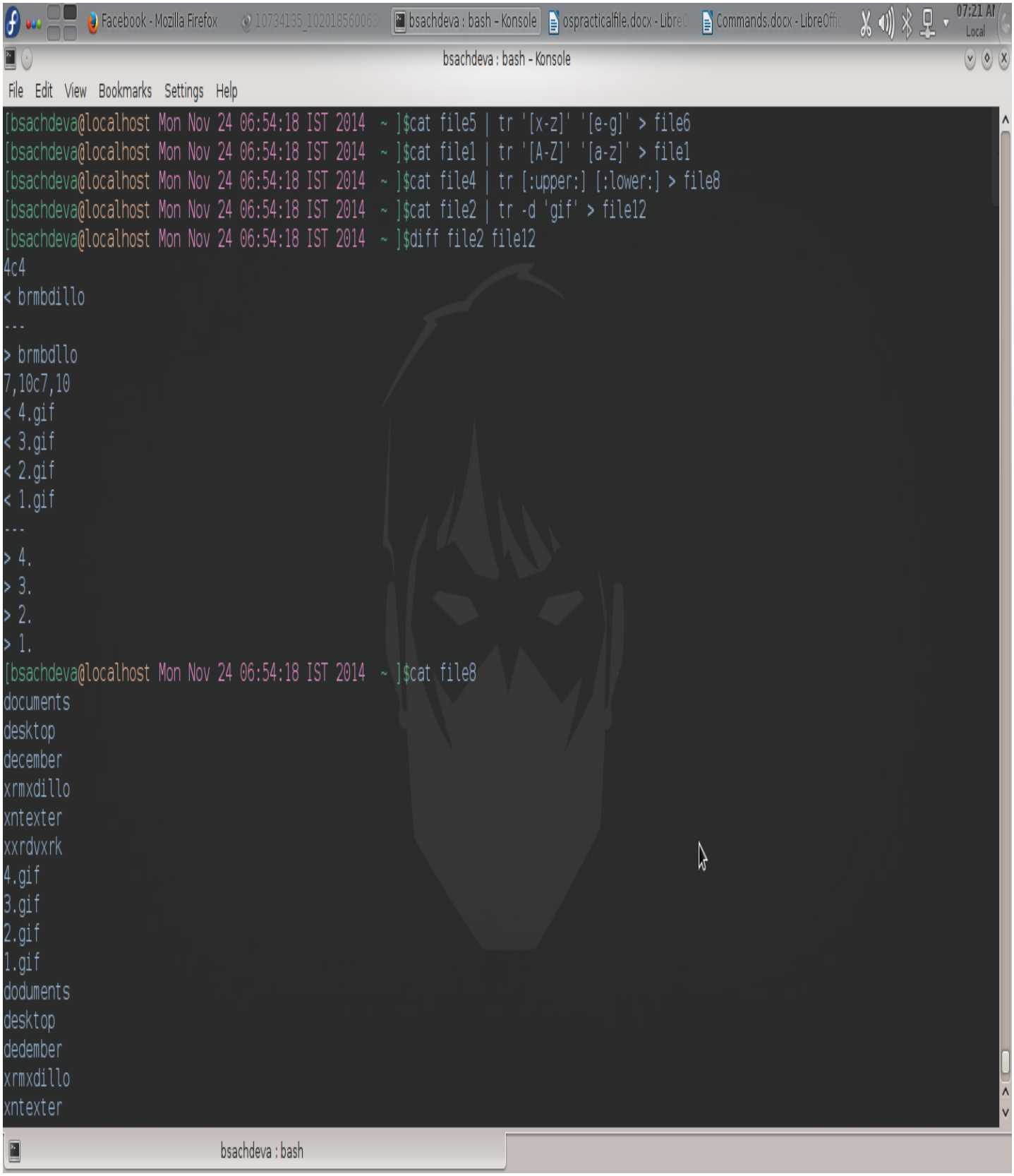
The quotation marks are necessary for tr to treat the argument as a string. If they are not used, everything in the argument is instead treated as individual characters. Thus, if the above example were rewritten without the quotation marks, it would remove every instance of the letters s, o, f and t. Interestingly, the quotation marks cannot be used to treat arguments as strings when not using the -d option.

Among the few remaining options is -c, which causes tr to work on the complement of the specified characters, that is, on the characters that are not in the given set.

tr contains much of most basic functionality of the command line program sed, which is used to perform basic editing on streams of text supplied by a pipe. However, it often advantageous to use tr instead of sed because the former is simpler and requires less typing and because it is easier to incorporate into scripts.


Facebook - Mozilla Firefox
bsachdeva : bash - Konsole
ospracticalfile.docx - Libre
Commands.docx - LibreOffi
07:13 AM
Local
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$tr a b
hamlet
hbmlet
bald
bbld
^C
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$tr a b < file1
Documents
Desktop
December
brmbdillo
bntebter
bbrdvbrk
4.gif
3.gif
2.gif
1.gif
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$tr a b < file1 >file2
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file
file1 file2 file3 file4
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file1
Documents
Desktop
December
armadillo
anteater
aardvark
4.gif
3.gif
2.gif
1.gif
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$tr c d < file1 >> file1
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file1
bsachdeva : bash


Facebook - Mozilla Firefox
bsachdeva : bash - Konsole
ospracticalfile.docx - LibreO
Commands.docx - LibreOffic
07:16 AM
Local
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
4.gif
3.gif
2.gif
1.gif
Doduments
Desktop
Dedember
armadillo
anteater
aardvark
4.gif
3.gif
2.gif
1.gif
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file1 | tr a x > file4
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file1 | tr '[efg]' '[xyz]' > file5
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file5
Documxnts
Dxsktop
Dxcxmbxr
armadillo
antxatxr
aardvark
4.ziy
3.ziy
2.ziy
1.ziy
Dodumxnts
Dxsktop
Dxdxmbxr
armadillo
antxatxr
aardvark
4.ziy
3.ziy
bsachdeva : bash


Facebook - Mozilla Firefox
bsachdeva : bash - Konsole
ospracticalfile.docx - Libre
Commands.docx - LibreOffic
07:21 AM
Local
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file5 | tr '[x-z]' '[e-g]' > file6
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file1 | tr '[A-Z]' '[a-z]' > file1
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file4 | tr [:upper:] [:lower:] > file8
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file2 | tr -d 'gif' > file12
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$diff file2 file12
4c4
< brmbdillo

> brmbdllo
7,10c7,10
< 4.gif
< 3.gif
< 2.gif
< 1.gif

> 4.
> 3.
> 2.
> 1.
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file8
documents
desktop
december
xrmxdillo
xntexter
xxrdvxrk
4.gif
3.gif
2.gif
1.gif
doduments
desktop
dedember
xrmxdillo
xntexter
bsachdeva : bash

The free Command

The free command provides information about unused and used memory and swap space on any computer running Linux or another Unix-like operating system. Memory consists of mainly of random access memory (RAM) chips that have been built into multi-chip modules that are, in turn, plugged into slots on the motherboard (i.e., the main circuit board on a personal computer or workstation). Swap space is is a portion of a hard disk drive (HDD) that is used to simulate additional main memory (i.e., which is used for virtual memory).

SYNOPSIS

free [options]

free accepts no arguments (i.e., input data) and is commonly used without any options. When used with no options, free presents a small table containing six columns and three rows of data, all expressed in kilobytes.

The first row, labeled Mem, displays physical memory utilization, including the amount of memory allocated to buffers and caches. A buffer, also called buffer memory, is usually defined as a portion of memory that is set aside as a temporary holding place for data that is being sent to or received from an external device, such as a HDD, keyboard, printer or network.

The second line of data, which begins with -/+ buffers/cache, shows the amount of physical memory currently devoted to system buffer cache. This is particularly meaningful with regard to application programs, as all data accessed from files on the system that are performed through the use of read() and write() system calls1 pass through this cache. This cache can greatly speed up access to data by reducing or eliminating the need to read from or write to the HDD or other disk.

The third row, which begins with Swap, shows the total swap space as well as how much of it is currently in use and how much is still available.

Several options are available to change the unit of display for free from its default kilobytes, including -b for bytes, -m for megabytes and -g for gigabytes. Of these, -m is usually the most useful. Thus, for example, to show all of the data in megabytes, the following would be used:

free -m

free will report slightly less memory as being in a computer than the actual total. This is mostly because the kernel (i.e., the core of the operating system) cannot be swapped out (i.e., the kernel always remains in main memory while the computer is in operation), and thus the memory that it occupies can never be freed. There can also be regions of memory that are reserved for other purposes, according to the specific system architecture.

The -t option instructs free to additionally display a fourth line of data containing the totals for physical memory and swap space.

The -s option followed by an integer tells free to keep providing a new, updated output at regular intervals for which the integer indicates the number of seconds. This scrolling output can be terminated by simultaneously pressing the CONTROL and c keys. Thus, for example, the following would provide new data every five seconds and display the output in megabits:

free -ms 5

An alternative to using free with its -s option would be to run it using the watch command, which by default runs a program provided to it as an argument every two seconds after first temporarily clearing the screen. This can make it easier to see changes as they occur because there is no scrolling. The program can be terminated and the screen returned to its former state by using the CONTROL and c key combination. Thus, for example, to display memory utilization every two seconds, the following would be used:

watch free

This command can be made even more useful by using watch's -d (i.e., difference) option, which highlights changes in output, and its -n option followed by the number one to increase the frequency of reports to one per second as follows:

watch -n 1 -d free

More detailed information about total memory and current memory usage can be obtained by reading the proc/meminfo file directly. This can be accomplished, for example, with the cat command (which is commonly used to read text files) as follows:

cat /proc/meminfo

DESCRIPTION

free displays the total amount of free and used physical and swap memory in the system, as well as the buffers used by the kernel. The shared memory column represents either the MemShared value (2.4 series kernels) or the Shmem value (2.6 series kernels and later) taken from the /proc/meminfo file. The value is zero if none of the entries is exported by the kernel.

-b, --bytes

Display the amount of memory in bytes.

-k, --kilo

Display the amount of memory in kilobytes. This is the default.

-m, --mega

Display the amount of memory in megabytes.

-g, --giga

Display the amount of memory in gigabytes.

--tera

Display the amount of memory in terabytes.

-h, --human

Show all output fields automatically scaled to shortest three digit unit and display the units of print out. Following units are used.

B = bytes

K = kilos

M = megas

G = gigas

T = teras

If unit is missing, and you have petabyte of RAM or swap, the number is in terabytes and columns might not be aligned with header.

-c, --count count

Display the result count times. Requires the -s option.

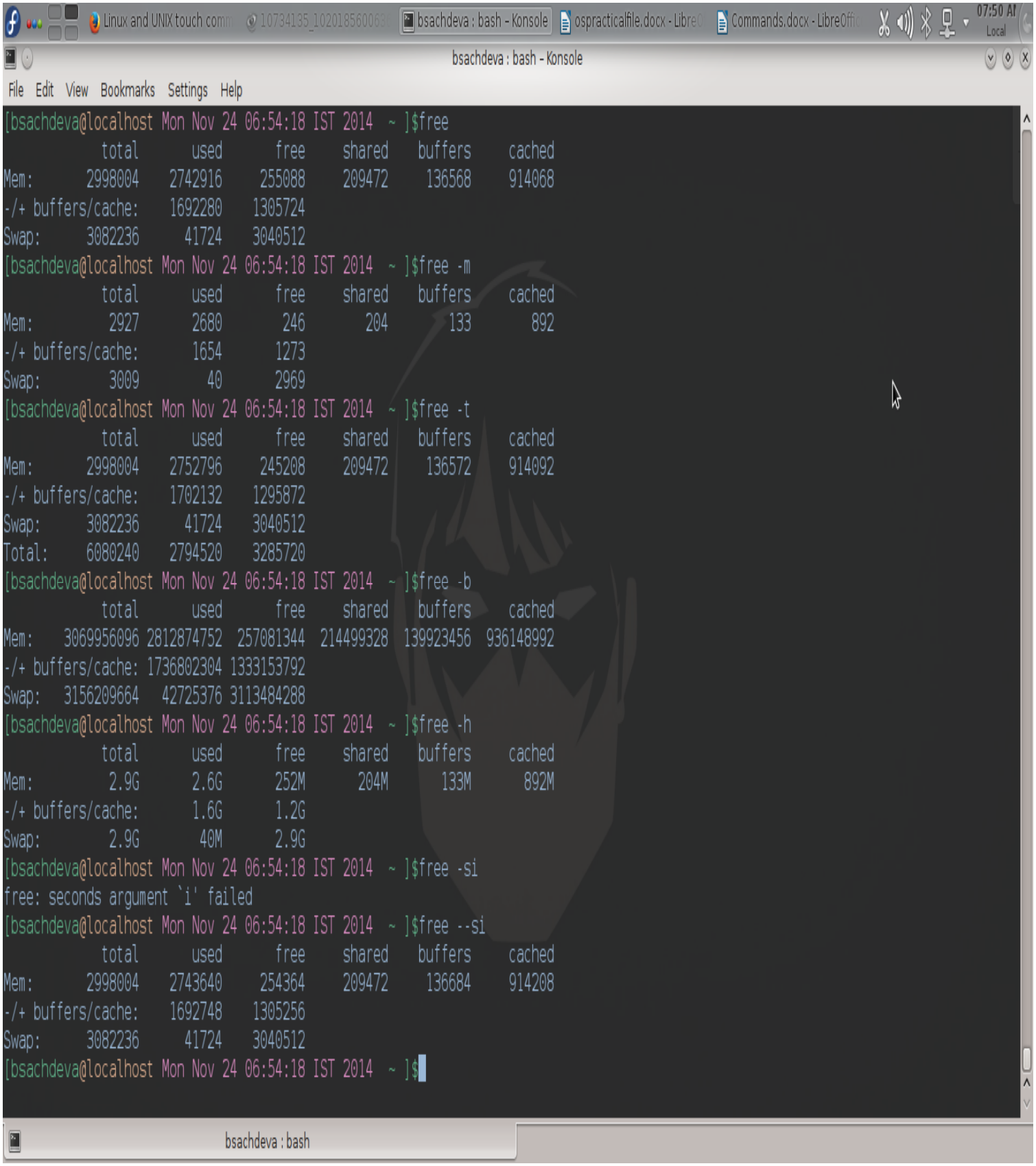
-s, --seconds seconds

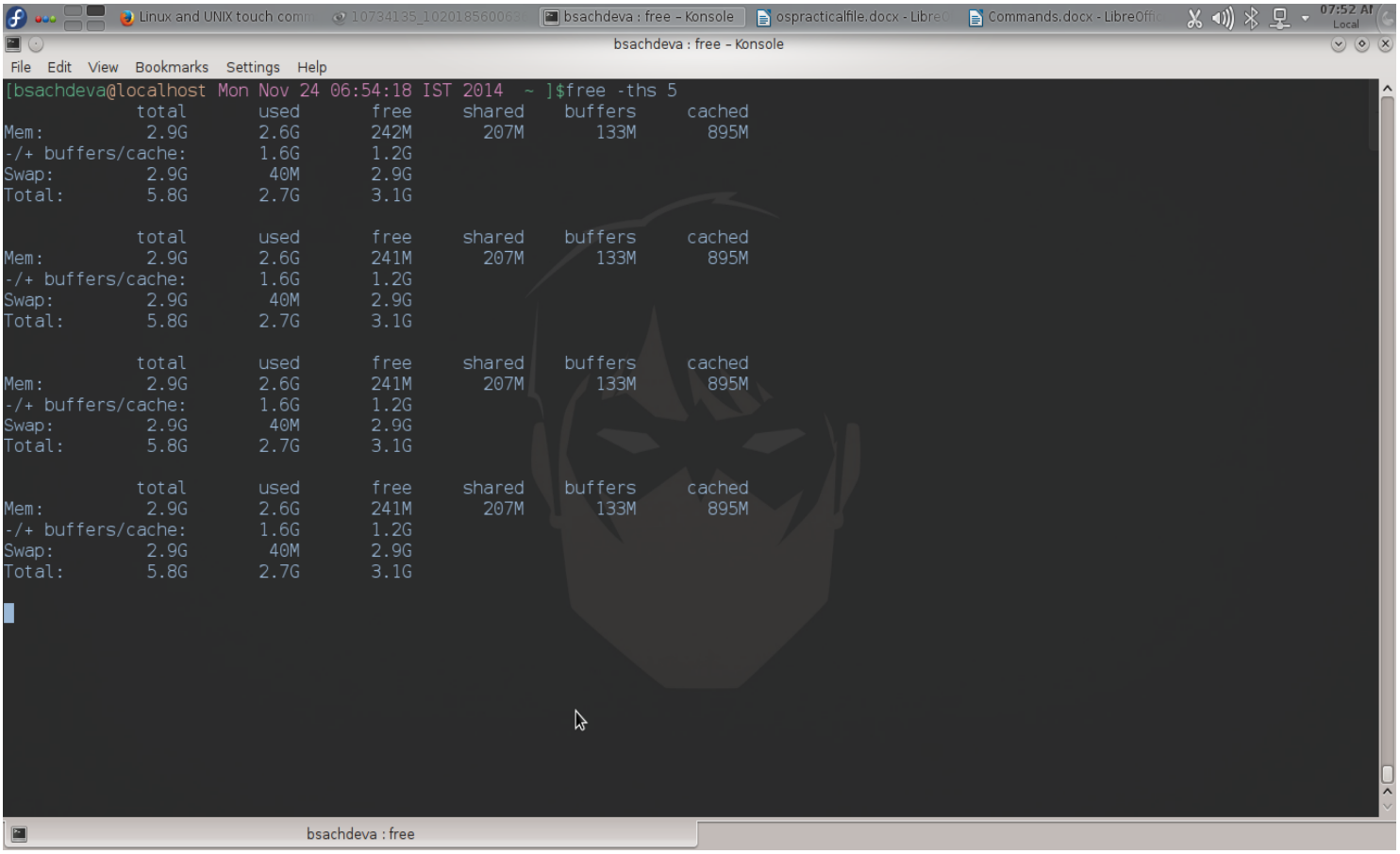
Continuously display the result delay seconds apart. You may actually specify any floating point number for delay.

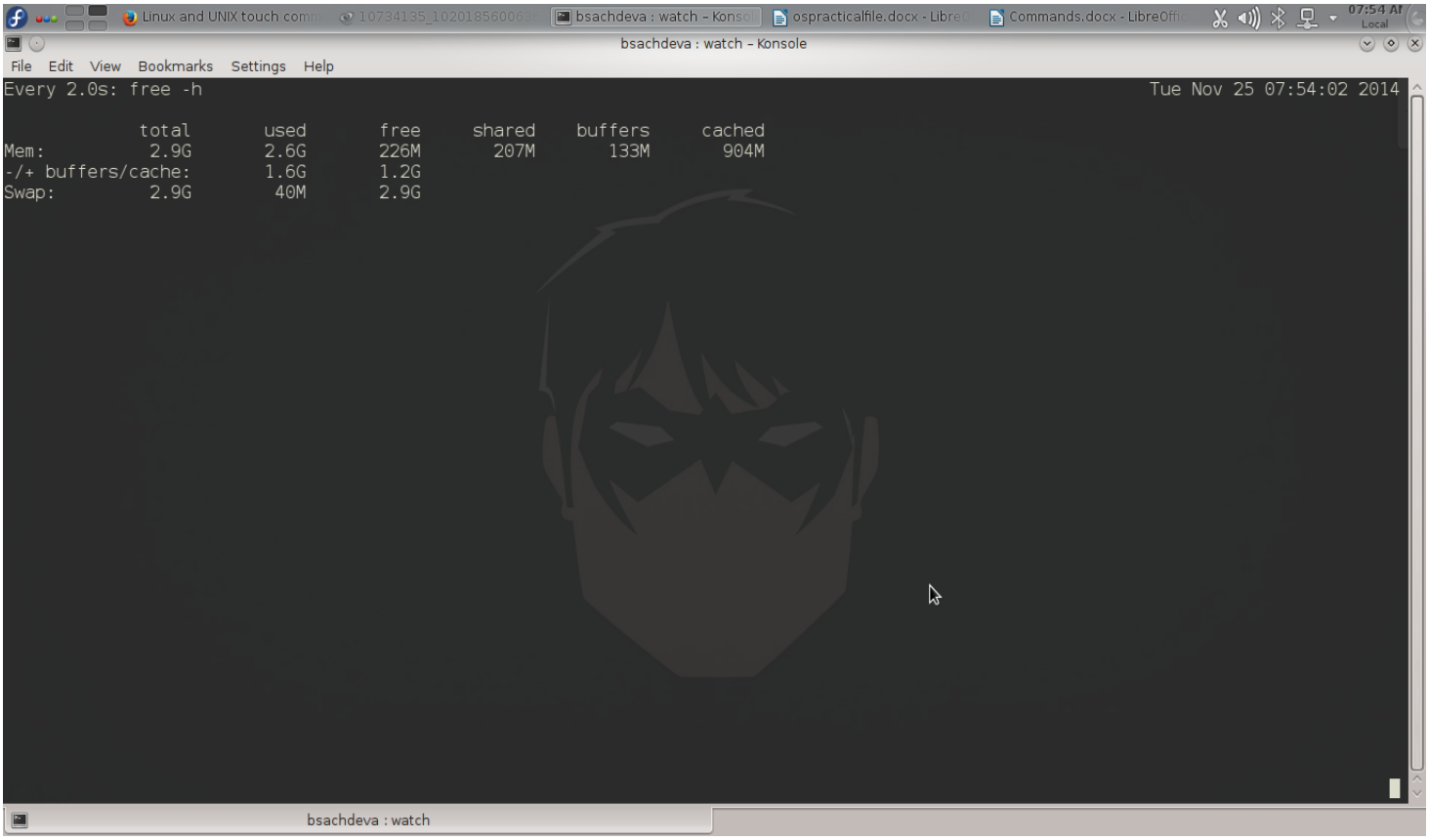
--si Use power of 1000 not 1024.

-t, --total

Display a line showing the column totals.


Linux and UNIX touch com
bsachdeva : bash - Konsole
ospracticalfile.docx - Libre
Commands.docx - LibreOffi
07:50 AM
Local
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$free
total used free shared buffers cached
Mem: 2998004 2742916 255088 209472 136568 914068
-/+ buffers/cache: 1692280 1305724
Swap: 3082236 41724 3040512
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$free -m
total used free shared buffers cached
Mem: 2927 2680 246 204 133 892
-/+ buffers/cache: 1654 1273
Swap: 3009 40 2969
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$free -t
total used free shared buffers cached
Mem: 2998004 2752796 245208 209472 136572 914092
-/+ buffers/cache: 1702132 1295872
Swap: 3082236 41724 3040512
Total: 6080240 2794520 3285720
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$free -b
total used free shared buffers cached
Mem: 3069956096 2812874752 257081344 214499328 139923456 936148992
-/+ buffers/cache: 1736802304 1333153792
Swap: 3156209664 42725376 3113484288
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$free -h
total used free shared buffers cached
Mem: 2.9G 2.6G 252M 204M 133M 892M
-/+ buffers/cache: 1.6G 1.2G
Swap: 2.9G 40M 2.9G
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$free -si
free: seconds argument `i' failed
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$free --si
total used free shared buffers cached
Mem: 2998004 2743640 254364 209472 136684 914208
-/+ buffers/cache: 1692748 1305256
Swap: 3082236 41724 3040512
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$
bsachdeva : bash


bsachdeva : free – Konsole
bsachdeva : free – Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$free -ths 5
total used free shared buffers cached
Mem: 2.9G 2.6G 242M 207M 133M 895M
-/+ buffers/cache: 1.6G 1.2G
Swap: 2.9G 40M 2.9G
Total: 5.8G 2.7G 3.1G
total used free shared buffers cached
Mem: 2.9G 2.6G 241M 207M 133M 895M
-/+ buffers/cache: 1.6G 1.2G
Swap: 2.9G 40M 2.9G
Total: 5.8G 2.7G 3.1G
total used free shared buffers cached
Mem: 2.9G 2.6G 241M 207M 133M 895M
-/+ buffers/cache: 1.6G 1.2G
Swap: 2.9G 40M 2.9G
Total: 5.8G 2.7G 3.1G
total used free shared buffers cached
Mem: 2.9G 2.6G 241M 207M 133M 895M
-/+ buffers/cache: 1.6G 1.2G
Swap: 2.9G 40M 2.9G
Total: 5.8G 2.7G 3.1G
bsachdeva : free


bsachdeva : watch – Konsole
bsachdeva : watch – Konsole
File Edit View Bookmarks Settings Help
Every 2.0s: free -h
Tue Nov 25 07:54:02 2014
total used free shared buffers cached
Mem: 2.9G 2.6G 226M 207M 133M 904M
-/+ buffers/cache: 1.6G 1.2G
Swap: 2.9G 40M 2.9G
bsachdeva : watch

The cut command

Cut command in unix (or linux) is used to select sections of text from each line of files. You can use the cut command to select fields or columns from a line by specifying a delimiter or you can select a portion of text by specifying the range or characters. Basically the cut command slices a line and extracts the text.

SYNOPSIS

cut OPTION... [FILE]...

We will see the usage of cut command by considering the below text file as an example
$ cat file.txt
unix or linux os
is unix good os
is linux good os

The cut command can be used to print characters in a line by specifying the position of the characters. To print the characters in a line, use the -c option in cut command

$cut -c4 file.txt
x
u
l
The above cut command prints the fourth character in each line of the file. You can print more than one character at a time by specifying the character positions in a comma separated list as shown in the below example

$cut -c4,6 file.txt
xo
ui
ln
This command prints the fourth and sixth character in each line.

You can print a range of characters in a line by specifying the start and end position of the characters.

$cut -c4-7 file.txt
x or
unix
linu
The above cut command prints the characters from fourth position to the seventh position in each line. To print the first six characters in a line, omit the start position and specify only the end position.

$cut -c-6 file.txt

```
unix o
is uni
is lin
```

To print the characters from tenth position to the end, specify only the start position and omit the end position.

```
$cut -c10- file.txt
inux os
ood os
good os
```

If you omit the start and end positions, then the cut command prints the entire line.

```
$cut -c- file.txt
```

You can use the cut command just as awk command to extract the fields in a file using a delimiter. The -d option in cut command can be used to specify the delimiter and -f option is used to specify the field position.

```
$cut -d ' ' -f2 file.txt
or
unix
linux
```

This command prints the second field in each line by treating the space as delimiter. You can print more than one field by specifying the position of the fields in a comma delimited list.

```
$cut -d' ' -f2,3 file.txt
or linux
unix good
linux good
```

The above command prints the second and third field in each line.

Note: If the delimiter you specified is not exists in the line, then the cut command prints the entire line. To suppress these lines use the -s option in cut command.

You can print a range of fields by specifying the start and end position.

```
$cut -d' ' -f1-3 file.txt
```

The above command prints the first, second and third fields. To print the first three fields, you can ignore the start position and specify only the end position.

$cut -d' ' -f-3 file.txt
To print the fields from second fields to last field, you can omit the last field position.

$cut -d' ' -f2- file.txt
cut command to display the first field from /etc/passwd file

The /etc/passwd is a delimited file and the delimiter is a colon (:). The cut command to display the first field in /etc/passwd file is

$cut -d':' -f1 /etc/passwd

The input file contains the below text

$ cat filenames.txt
logfile.dat
sum.pl
add_int.sh

Using the cut command extract the portion after the dot.

First reverse the text in each line and then apply the command on it.

$rev filenames.txt | cut -d'.' -f1

DESCRIPTION

-b, --bytes=LIST

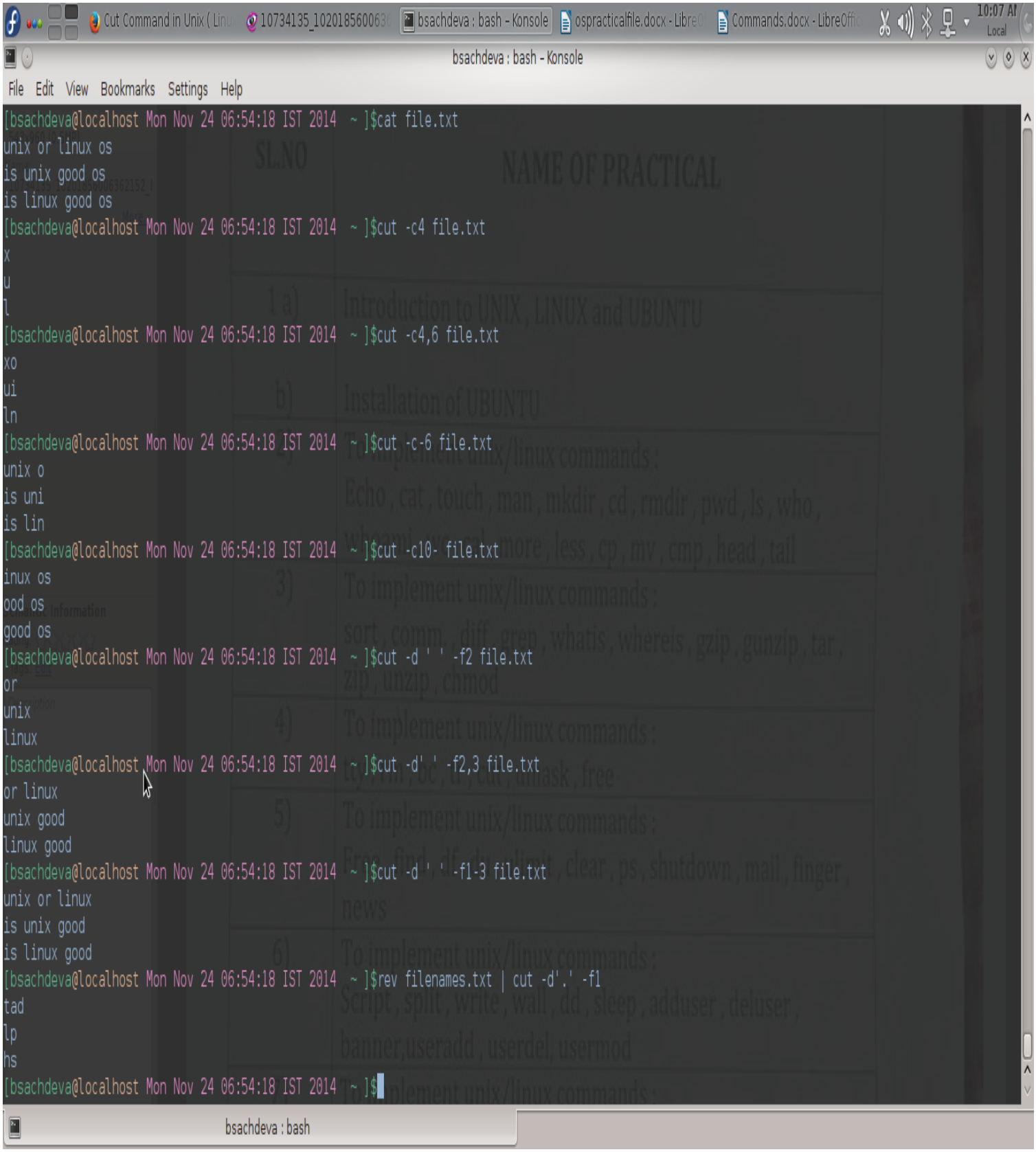
select only these bytes

-c, --characters=LIST

select only these characters

-d, --delimiter=DELIM

use DELIM instead of TAB for field delimiter


Cut Command in Unix (Linu
10734135_102018560063
bsachdeva : bash - Konsole
ospracticalfile.docx - Libre
Commands.docx - LibreOffic
10:07 AM
Local
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cat file.txt
unix or linux os
is unix good os
is linux good os
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cut -c4 file.txt
x
u
l
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cut -c4,6 file.txt
xo
ui
ln
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cut -c-6 file.txt
unix o
is uni
is lin
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cut -c10- file.txt
inux os
ood os
good os
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cut -d ' ' -f2 file.txt
or
unix
linux
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cut -d' ' -f2,3 file.txt
or linux
unix good
linux good
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$cut -d ' ' -f1-3 file.txt
unix or linux
is unix good
is linux good
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$rev filenames.txt | cut -d'.' -f1
tad
lp
hs
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 ~]$
bsachdeva : bash

The bc command

An arbitrary precision calculator language

SYNOPSIS

bc *optionsfile...*

Standard functions supported by bc

length (expression)
The value of the length function is the number of significant digits in the expression.
read ( )
Read a number from the standard input, regardless of where the function occurs. Beware, this can cause problems with the mixing of data and program in the standard input. The best use for this function is in a previously written program that needs input from the user, but never allows program code to be input from the user.

scale (expression)
The number of digits after the decimal point in the expression.

sqrt (expression)
The square root of the expression.

Most standard math expressions are of course supported: + - / * % ^

++ var
increment the variable by one and set the new value as the result of the expression.

var ++
The result of the expression is the value of the variable and the variable is then incremented by one.

-- var
decrement the variable by one and set the new value as the result of the expression.

var --
The result of the expression is the value of the variable and the variable is then decremented by one.

(expr)
Parenthesis alter the standard precedence to force the evaluation of an expression.

var = expr
The variable var is assigned the value of the expression.

Relational expressions and Boolean operations are also legal, look at the full bc man page for more

DESCRIPTION

file A file containing the calculations/functions to perform.

May be piped from standard input

-i, --interactive

Force interactive mode.

-l, --mathlib

Define the standard math library.

-w, --warn

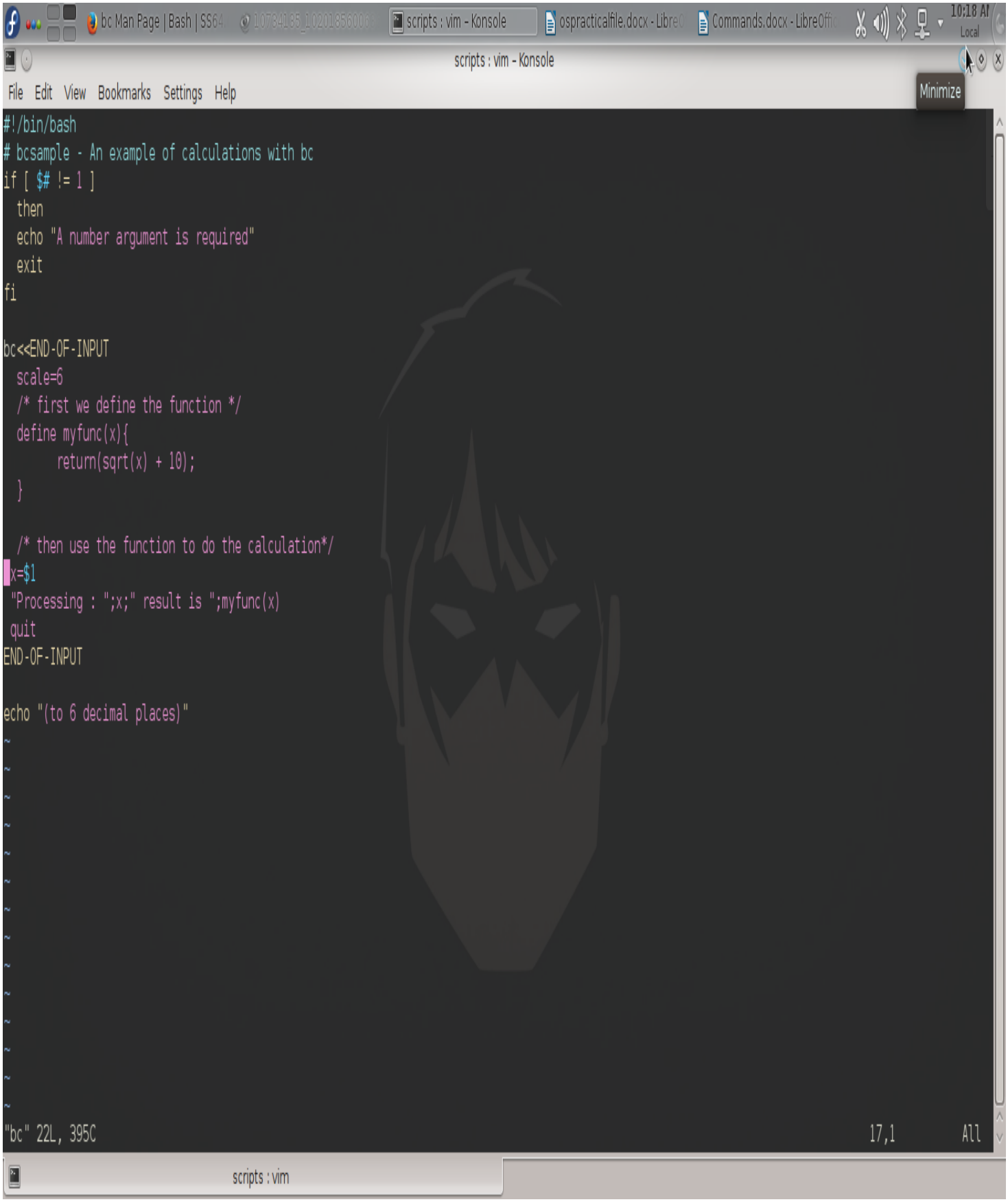
Give warnings for extensions to POSIX bc.

-s, --standard

Process exactly the POSIX bc language.

-q, --quiet

Do not print the normal GNU bc welcome.


bc Man Page | Bash | SS64
scripts : vim - Konsole
ospracticalfile.docx - LibreO
Commands.docx - LibreOffic
10:18 AI
Local
scripts : vim - Konsole
File Edit View Bookmarks Settings Help
Minimize
#!/bin/bash
bcsample - An example of calculations with bc
if [$# != 1]
then
echo "A number argument is required"
exit
fi
bc<<END-OF-INPUT
scale=6
/* first we define the function */
define myfunc(x){
return(sqrt(x) + 10);
}
/* then use the function to do the calculation*/
x=$1
"Processing : ";x;" result is ";myfunc(x)
quit
END-OF-INPUT
echo "(to 6 decimal places)"
"bc" 22L, 395C
17,1
All
scripts : vim

The tty command

Print file name of terminal on standard input, print the file name of the terminal connected to standard input. It prints `not a tty' if standard input is not a terminal.

SYNOPSIS

```
      tty [option]...
```

DESCRIPTION

```
   -s
   --silent
   --quiet
           Print nothing; only return an exit status.

 Exit status:

     0 if standard input is a terminal
     1 if standard input is not a terminal
     2 if given incorrect arguments
     3 if a write error occurs
```

The df Command

The df (i.e., disk free) command reports the amount of space used and available on currently mounted (i.e., logically attached to the system) filesystems. A filesystem is a hierarchy of directories. Although everything mounted on a computer running Linux or another Unix-like operating system resides logically in the root directory (i.e., the top directory in the hierarchy of directories on a computer), on a typical installation some filesystems will be physically located on different partitions (i.e., a logically independent section of a hard disk drive) or on other storage devices such as a CDROM, floppy disk or a USB (universal serial bus) key drive, or even on another computer located elsewhere on a network.

df is commonly used to show not only the space available and used but also what filesystems are mounted, when capacity has been reached on a partition or other device, the number of inodes still available, and whether there is sufficient space on a partition to upgrade an existing program or to install a new program.

SYNOPSIS

df [option(s)] [device(s)]

The items in square brackets are optional. When used with no options and no arguments (i.e., input files), df generates a table that lists for each device its name, its size in one-kilobyte blocks, the number of kilobyte blocks used, the number of kilobyte blocks available, the percentage of blocks used and the mount point (i.e., the directory in which it is mounted).

When a device is provided as an argument, df will report data only for the filesystem physically

residing on that device. If multiple devices are provided, the data will be reported for each listed device. Thus, for example, the following will provide information only for the partition (or other device) that contains the root directory:

```
df /
```

A device can be provided as an argument by using either its device name or its absolute pathname (i.e., its full path of directories relative to the root directory). Thus, for example, if the root directory (which is represented by a forward slash) resides on the third partition of the HDD (or on the third partition of the first HDD on a computer with multiple HDDs), whose device name is /dev/hda3, then the same result would be obtained by using the following:

```
df /dev/hda3
```

For many users, df's most useful option is -h, which tells it to express its output in human readable form. That is, it reports device sizes in megabytes and gigabytes as appropriate and appends the data with the letters M and G, respectively. The -m option tells df to report its data in one-megabyte (i.e., 1,048,576 bytes) blocks, and the -k option tells it to use one-kilobyte blocks (the default, but which is available as an option anyway).

The -i option instructs df to list inode usage instead of block usage. An inode is a data structure on a filesystem on a Unix-like operating system that stores all the information about a named file, directory or other filesystem object except its name and its actual data (i.e., all of its metadata except for its name). A data structure is an efficient way of storing data.

The -a (i.e., all) option tells df to include in its output filesystems that have a size of zero blocks, which are omitted by default. Such filesystems are typically special-purpose pseudo-filesystems, such as those for automatic device mounting use.

The -T option instructs df to add to its report the type of each filesystem listed (e.g., ext2, ext3, vfat or iso9660). The types reported are the same as those that can be included or excluded with the -t (i.e., type) and -x (i.e., exclude) options, respectively.

The --help option displays a brief help message and then exits. The --version option reports which version of df is currently installed. In contrast to many command line (i.e., all-text mode) programs, the -v option does not provide version information; it is merely ignored and does not produce an error message.

df's output data is not precise, and it can conflict with that provided by the du (i.e., disk usage) command, whose results likewise might not be exact. However, it is still a very useful command, and in many cases it is faster and more convenient than du.

DESCRIPTION

Show information about the file system on which each FILE resides, or all file systems by default.

Mandatory arguments to long options are mandatory for short options too.

-a, --all

include dummy file systems

-B, --block-size=SIZE

scale sizes by SIZE before printing them. E.g., '-BM' prints sizes in units of 1,048,576 bytes. See SIZE format below.

--direct

show statistics for a file instead of mount point

--total

produce a grand total

-h, --human-readable

print sizes in human readable format (e.g., 1K 234M 2G)

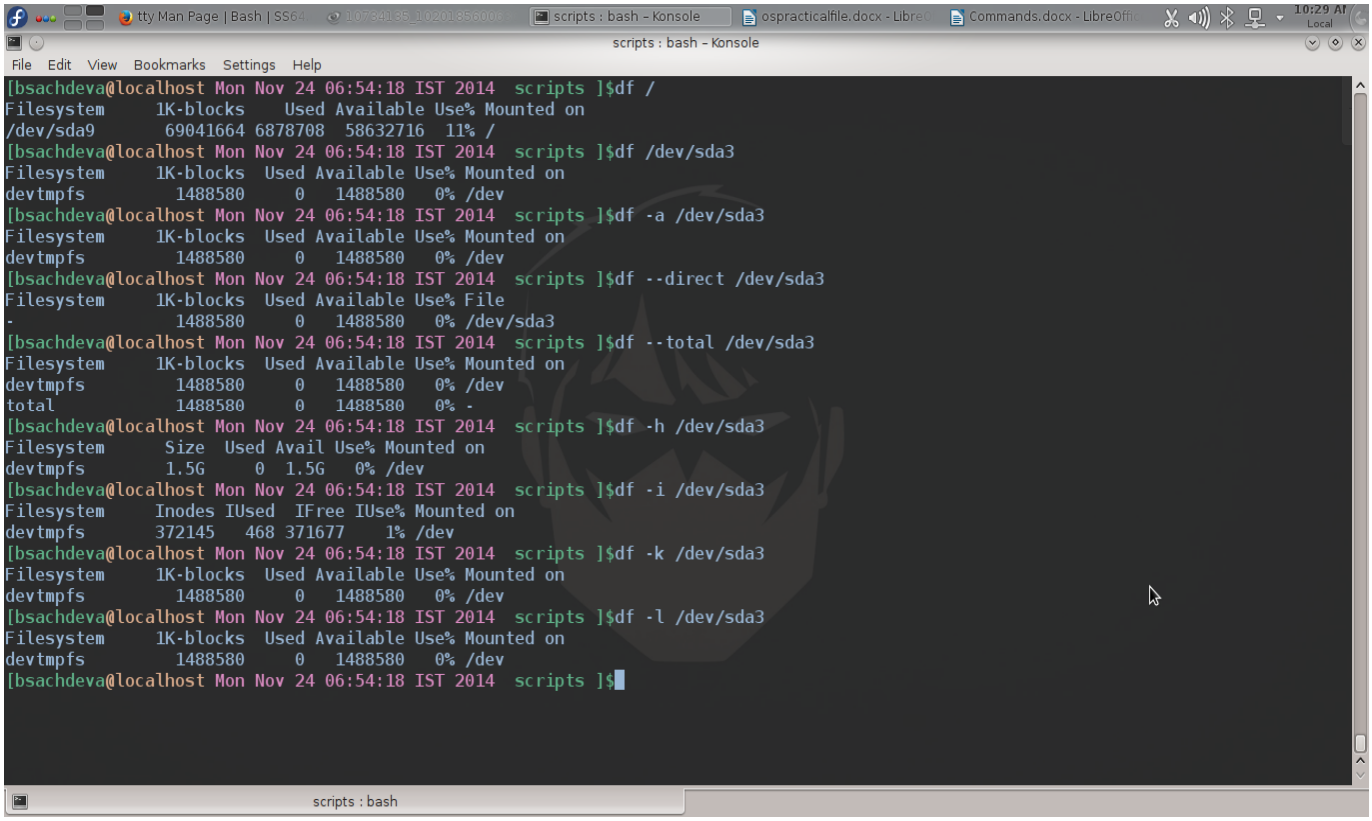
-i, --inodes

list inode information instead of block usage

-k like --block-size=1K

-l, --local

limit listing to local file systems



The du Command

The du (i.e., disk usage) command reports the sizes of directory trees inclusive of all of their contents and the sizes of individual files. This makes it useful for tracking down space hogs, i.e., directories and files that consume large or excessive amounts of space on a hard disk drive (HDD) or other storage media. A directory tree is a hierarchy of directories that consists of a single directory, called the parent directory or top level directory, and all levels of its subdirectories (i.e., directories within a directory). Any directory can be regarded as being the start of its own directory tree, at least if it contains subdirectories. Thus, a typical computer contains a large number of directory trees.

du is commonly employed by system administrators as a supplement to automated monitoring and notification programs that help prevent key directories and partitions (i.e., logically independent sections of a HDD) from becoming full. Full, or even nearly full, directories and partitions can cause a system to slow down, prevent users from logging in and even result in a system crash. Although visually identifying heavy consumers of disk space can be practical if there are relatively few users on a system, it is clearly not efficient for large systems with hundreds or thousands of users.

A minor limitation of du is the fact that the sizes of directories and files it reports are approximations, not exact numbers, and there is frequently a small discrepancy between these sizes and the sizes reported by other commands. However, this rarely detracts from its usefulness.

Also, du can only be used to estimate space consumption for directories and files for which the user has

reading permission. Thus, an ordinary user would generally not be able to use du to determine space consumption for files or directories belonging to other users, including those belonging to the root account (i.e., the system administrator). However, as du is used mainly by system administrators, this is usually not a problem.

SYNOPSIS

du [options] [directories and/or files]

The items in the square brackets are optional. When used with no options or arguments (i.e., names of directories or files), du lists the names and space consumption of each of the directories (including all levels of subdirectories) in the directory tree that begins with the current directory (i.e., the directory in which the user is currently working). The space consumption of any directory consists of the space occupied by all of the files in it and all of its subdirectories at all levels inclusive of all of the files in them. A final line at the end of the report gives the total space consumption for the directory tree.

du can provide information about any directory trees or files on the system whose names are given as arguments. For example, the following will report the names and sizes for each directory in the directory tree that begins with a directory named directory2 that resides in a directory named directory1, which, in turn, is located in the current directory:

du directory1/directory2

Likewise, the following will report the sizes of the two files named file1 and file2 that are located in the /sbin directory (which contains executable programs):

du /sbin/file1 /sbin/file2

du can accept any number of arguments, and they can be any combination of files and directories. When there are multiple arguments, no grand total is provided by default, although a total is still provided for each argument.

Options

As is the case with most commands on Linux and other Unix-like operating systems, du has a number of options, a few of which are commonly used. The options can vary somewhat according to the particular operating system and the version of du.

One of the most useful options is -h (i.e., human readable), which can make the output easier to read by displaying it in kilobytes (K), megabytes (M) and gigabytes (G) rather than just in the default kilobytes. Thus, the following command can be used to show the sizes of all the subdirectories in the current directory as well as the total size of the current directory, all formatted with the appropriate K, M or G:

du -h

The -s (for suppress or summarize) option tells du to report only the total disk space occupied by a directory tree and to suppress individual reports for its subdirectories. Thus, for example, the following would provide the total disk space occupied by the current directory in an easy-to-read format:

```
du -sh
```

The output is the same as the last line of a report issued by du with only the -h option.

The -a (i.e., all) option tells du to report not just the total disk usage for each directory at every level in a directory tree but also to report the space consumption for each individual file anywhere within the tree. Thus, for example, the following would list the name and size of every directory and file in the /etc directory (which contains system configuration files) for which the user has reading permission:

```
du -a /etc
```

A somewhat similar report is provided by using the star (*) wildcard, which will match any character or characters. For example, the following command would list the sizes of all directories that are in the tree that begins with the current directory:

```
du *
```

However, the only files listed are those in the the parent directory, not those in its subdirectories. Also, no total for the directory tree as a whole is provided.

The use of the -s option and the star wildcard together would cause du to report the names and sizes of only the files and directories contained directly in the top level directory itself (and to not list the names of any of its subdirectories and the files in them). The size of each listed directory is, of course, inclusive of all of its files and subdirectories (including all of the files in them). For example, such a report about the directory tree beginning with the current directory would be provided by the following:

```
du -hs *
```

The wildcard can also be used to filter the output to list only those items whose names begin with, contain or end with certain characters or sequences of characters. For example, the following would report the names and sizes of all of the directories and files in the current directory whose names begin with the letter s as well as the names and sizes of all levels of subdirectories of those directories regardless of what their names begin with:

```
du -h s*
```

The -c option can be added to provide a grand total for all of the files and directories that are listed. In the case of the above example, this would be

```
du -hc s*
```

As another example of the use of the wildcard, the following command would report the name and size of each gif (one of the two most popular image formats) file in the current directory as well as a total for all of the gifs:

```
du -hc *.gif
```

Another useful option is --max-depth=, which instructs du to list its subdirectories and their sizes to any desired level of depth (i.e., to any level of subdirectories) in a directory tree. For example, the

following would cause du to list only the first tier (i.e., layer) of directories in the current directory and their sizes (inclusive of all of their contents, including those of their subdirectories):

```
du --max-depth=1
```

The total space consumption for the current directory tree will also be reported, and it will, of course, be the same regardless of the depth of the files listed.

Setting --max-depth= to zero tells du to not list any of the subdirectories within the selected directory, i.e., to list only report the size of the selected directory itself. The result is the same as using the -s option.

Using du With Filters

As is the case with other commands on Unix-like operating systems, du can be linked with pipes to filters to create powerful pipelines of commands. A filter is a (usually) small and specialized program that transforms data in some meaningful way.

For example, to arrange the output items according to size, du can be piped to the sort command, whose -n option tells it to list the output in numeric order with the smallest files first, as follows:

```
du | sort -n
```

As du will often generate more output than can fit on the monitor screen at one time, the output will fly by at high speed and be virtually unreadable. Fortunately, it is easy to display the output one screenful at a time by piping it to the less filter, for example,

```
du -h | less
```

The output of less can be advanced one screenful at a time by pressing the space bar, and it can be moved backward one screenful at a time by pressing the b key.

The output of du can likewise be piped to less after it has been passed through one or more other filters, for example,

```
du -h | sort -n | less
```

The grep filter can be used to search through du's output for any desired string (i.e., sequence of characters). Thus, for example, the following will provide a list of the names and sizes of directories and files in the current directory that contain the word linux:

```
du -ah | grep linux
```

One way in which du can be used to produce a list of (mostly) directories and files in a directory tree that are consuming large amounts of disk space is to use grep to search for all the lines that contain the upper case letter M (i.e., for megabytes) or G (for gigabytes), such as

```
du -ah | grep M
```

The only problem with this approach is that it will also select directories and files that contain an upper

case M or G in their names even if the file size is not measured in megabytes or gigabytes. (However, this problem could be overcome through the use of regular expressions, an advanced pattern matching technique).

Alternatives to du

There are several other ways of monitoring disk space consumption and reporting file sizes. Although very useful tools, they are generally not good substitutes for du.

Among them is the df command, which is likewise used by system administrators to monitor disk usage. However, unlike du, it can only show the space consumption on entire partitions, and it lacks du's fine-grained ability to track the space usage of individual directories and files.

du is not designed to show the space consumption of partitions. The closest that it can come is to show the sizes of the first tier of directories in the root directory (i.e., the directory which contains all other directories and which is represented by a forward slash), several of which may be on their own partitions (depending on how the system has been set up). This is accomplished by becoming the root user and issuing the following command:

```
du -h --max-depth=1 /
```

The ls (i.e., list) command can provide the sizes of individual files by using its -s option, and its -h option (which is similar to du's -h option) can be added to make the output easier to read. For example, the following would list the names and sizes of the files in the current directory:

```
ls -sh
```

Although the names of the first tier of directories within the current directory are also listed, the size data accompanying them does not represent their actual disk space consumption (i.e., inclusive of their contents). Nor does ls report the contents of any lower tiers of directories, unless such directories are specifically listed as arguments.

A convenient alternative for finding the sizes of files and directory trees when using a GUI (graphical user interface) is to click with the right mouse button on the icon (i.e., a small picture or symbol) for that item and then select Properties from the menu that appears. Although this is frequently sufficient, it does not provide the detailed control and reporting that du provides.

DESCRIPTION

-a, --all

write counts for all files, not just directories

('sparse') files, internal fragmentation, indirect blocks, and the like

-B, --block-size=SIZE

scale sizes by SIZE before printing them. E.g., '-BM' prints sizes in units of 1,048,576 bytes. See SIZE format below.

-b, --bytes

equivalent to '--apparent-size --block-size=1'

-c, --total

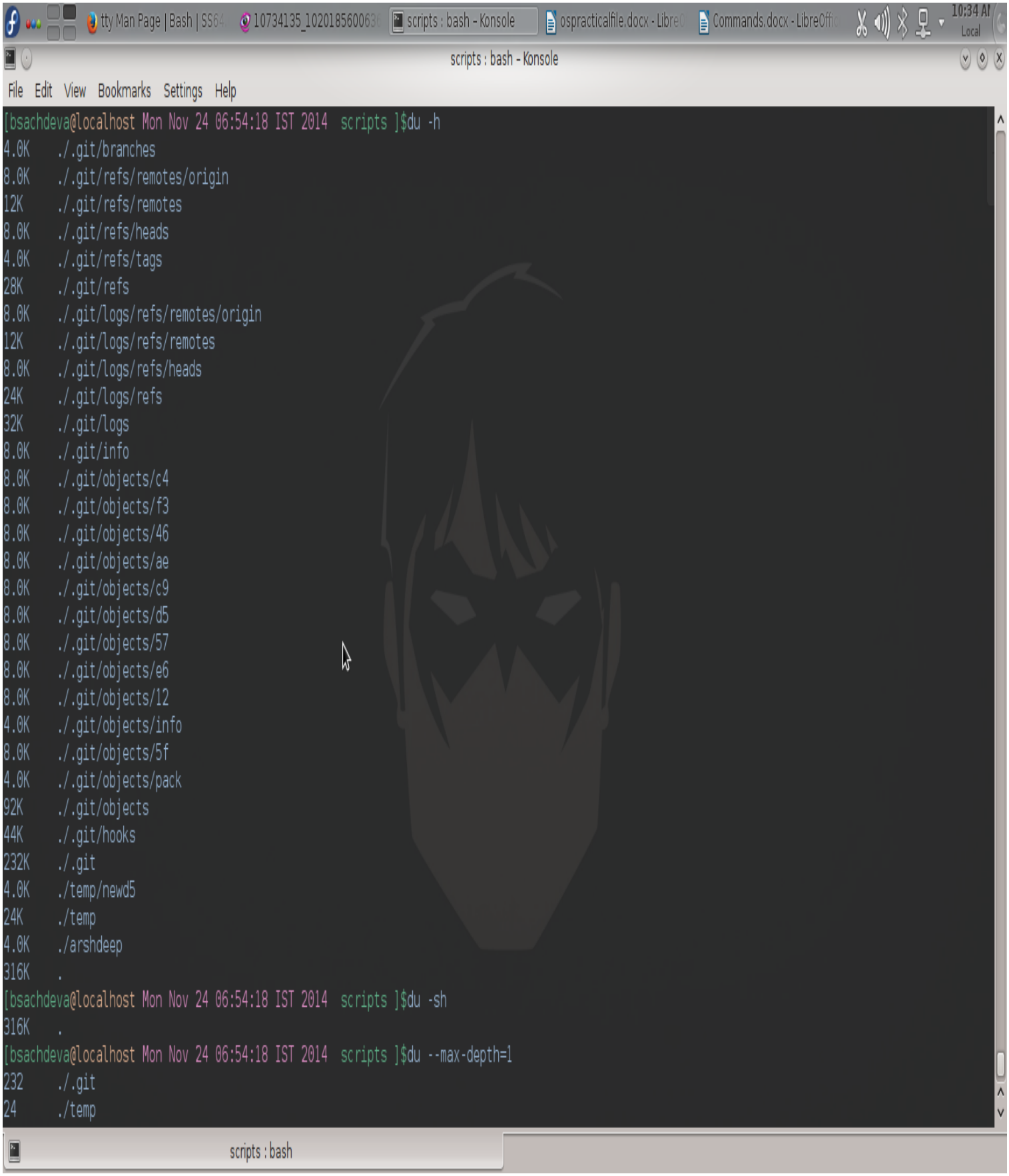
produce a grand total

-h, --human-readable

print sizes in human readable format (e.g., 1K 234M 2G)

-s, --summarize

display only a total for each argument.


scripts : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$du -h
4.0K ./.git/branches
8.0K ./.git/refs/remotes/origin
12K ./.git/refs/remotes
8.0K ./.git/refs/heads
4.0K ./.git/refs/tags
28K ./.git/refs
8.0K ./.git/logs/refs/remotes/origin
12K ./.git/logs/refs/remotes
8.0K ./.git/logs/refs/heads
24K ./.git/logs/refs
32K ./.git/logs
8.0K ./.git/info
8.0K ./.git/objects/c4
8.0K ./.git/objects/f3
8.0K ./.git/objects/46
8.0K ./.git/objects/ae
8.0K ./.git/objects/c9
8.0K ./.git/objects/d5
8.0K ./.git/objects/57
8.0K ./.git/objects/e6
8.0K ./.git/objects/12
4.0K ./.git/objects/info
8.0K ./.git/objects/5f
4.0K ./.git/objects/pack
92K ./.git/objects
44K ./.git/hooks
232K ./.git
4.0K ./temp/newd5
24K ./temp
4.0K ./arshdeep
316K .
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$du -sh
316K .
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$du --max-depth=1
232 ./.git
24 ./temp
scripts : bash

The clear Command

The clear command is used to remove all previous commands and output from consoles and terminal windows in Unix-like operating systems.

A console is an all-text mode user interface that occupies the entire screen of the display device and which does not sit on top of a graphical user interface (GUI). A terminal window is a text-only window in a GUI that emulates a console and which can be opened by clicking the appropriate icon (i.e., small image) or menu item.

Clear is one of the very few commands in Unix-like operating systems that accepts neither options nor arguments (i.e., input files). That is, it is only used as follows:

```
clear
```

After the clear command has been issued, all that remains on the display screen is the command prompt in the upper left hand corner. A command prompt, also referred to as a prompt, is a short text message at the start of a line that informs the user that the system is ready for the next command, data element or other input.

Removing previous commands and output can make it easier for users to focus on and understand subsequent commands and their output. It can also be useful in preventing the next user on a system from seeing what the previous user did.

clear first looks in the environment (i.e., a set of variables that tells how to run the system) to find the type of terminal that is being used, and then it looks in the terminfo database to determine how to empty the screen of all but the command prompt on a single line. The terminfo database describes the all-text mode terminals on the system by giving a set of their capabilities, specifying how to perform screen operations and specifying initialization sequences and padding requirements.

In Unix-like operating systems there are often multiple ways of performing the same task. For example, the same result as using the clear command can be achieved in the bash shell by simultaneously pressing the CONTROL and l (lower case L) keys. A shell is a program that provides the traditional, text-only user interface in consoles and terminal windows, and bash is the default shell on Linux.

The bash shell also allows a single line to be cleared by simultaneously pressing the CONTROL and 7 (number seven) keys. In addition, everything except the current line can be removed by simultaneously pressing the CONTROL and L (upper case L) keys.

The ps Command

The ps (i.e., process status) command is used to provide information about the currently running processes, including their process identification numbers (PIDs).

A process, also referred to as a task, is an executing (i.e., running) instance of a program. Every process is assigned a unique PID by the system.

SYNOPSIS

```
ps [options]
```

When ps is used without any options, it sends to standard output, which is the display monitor by default, four items of information for at least two processes currently on the system: the shell and ps. A shell is a program that provides the traditional, text-only user interface in Unix-like operating systems for issuing commands and interacting with the system, and it is bash by default on Linux. ps itself is a process and it dies (i.e., is terminated) as soon as its output is displayed.

The four items are labeled PID, TTY, TIME and CMD. TIME is the amount of CPU (central processing unit) time in minutes and seconds that the process has been running. CMD is the name of the command that launched the process.

TTY (which now stands for terminal type but originally stood for teletype) is the name of the console or terminal (i.e., combination of monitor and keyboard) that the user logged into, which can also be found by using the tty command. This information is generally only useful on a multi-user network.

A common and convenient way of using ps to obtain much more complete information about the processes currently on the system is to use the following:

```
ps -aux | less
```

The -a option tells ps to list the processes of all users on the system rather than just those of the current user, with the exception of group leaders and processes not associated with a terminal. A group leader is the first member of a group of related processes.

The -u option tells ps to provide detailed information about each process. The -x option adds to the list processes that have no controlling terminal, such as daemons, which are programs that are launched during booting (i.e., computer startup) and run unobtrusively in the background until they are activated by a particular event or condition.

As the list of processes can be quite long and occupy more than a single screen, the output of ps -aux can be piped (i.e., transferred) to the less command, which lets it be viewed one screen at a time. The output can be advanced one screen forward by pressing the SPACE bar and one screen backward by pressing the b key.

Among the information that ps -aux provides about each process is the user of the process, the PID, the percentage of CPU used by the process, the percentage of memory used by the process, VSZ (virtual

size in kilobytes), RSS (real memory size or resident set size in 1024 byte units), STAT (the process state code), the starting time of the process, the length of time the process has been active and the command that initiated the process. The process state codes include D, uninterrupted sleep; N, low priority; R, runnable (on run queue); S, sleeping; T, traced or stopped; Z, defunct (zombie).

In contrast to most commands, the hyphen preceding ps's options is optional, not mandatory. Thus, the following could be (and sometimes is) used in place of the above command:

```
ps aux | less
```

An alternative set of options for viewing all the processes running on a system is

```
ps -ef | less
```

The -e option generates a list of information about every process currently running. The -f option generates a listing that contains fewer items of information for each process than the -l option.

Among the columns displayed by ps -ef, UID contains the username of the account that owns the process (which is usually the same user that started the process) and STIME displays the time the process started, or the starting date if it started more than 24 hours ago.

The processes shown by ps can be limited to those belonging to any given user by piping the output through grep, a filter that is used for searching text. For example, processes belonging to a user with a username adam can be displayed with the following:

```
ps -ef | grep adam
```

The -l option generates a long listing, and when used together with the -e and -f options creates a table with 15 columns:

```
ps -efl
```

The additional columns of most interest are NI and SZ. The former shows the nice value of the process, which determines the priority of the process. The higher the value, the lower the priority. The default nice value is 0 on Linux systems.

The latter displays the size of the process in memory. The value of the field is the number of pages the process is occupying. On Linux systems a page is 4,096 bytes.

ps is most often used to obtain the PID of a malfunctioning process in order to terminate it with the kill command. For example, if the PID of a frozen or crashed program is found to be 1125, the following can usually terminate the process:

```
kill 1125
```

ps -ef or ps -efl can then be used to confirm that the process really has stopped. If it has not, then the more forceful -9 option should be used, i.e.,

```
kill -9 1125
```

The pstree command is similar to ps in that it can be used to show all of the processes on the system along with their PIDs. However, it differs in that it presents its output in a tree diagram that shows how processes are related to each other and in that it provides less detailed information about each process than does ps.

DESCRIPTION

-a Select all processes except both session leaders (see getsid(2)) and processes not associated with a terminal.

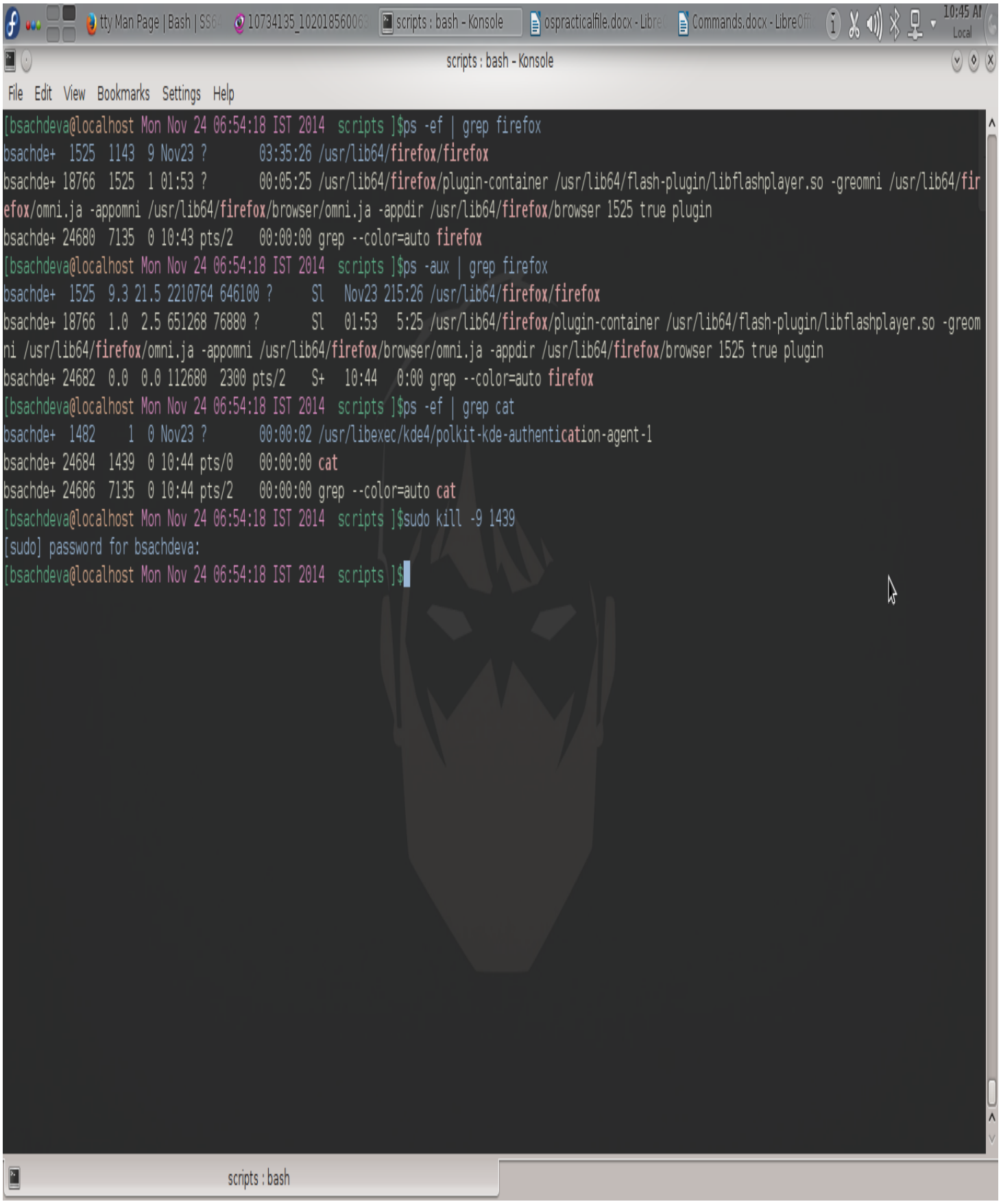
-e Select all processes. Identical to -A.

-x Lift the BSD-style "must have a tty" restriction, which is imposed upon the set of all processes when some BSD-style (without "-") options are used or when the ps personality setting is BSD-like. The set of processes selected in this manner is in addition to the set of processes selected by other means. An alternate description is that this option causes ps to list all processes owned by you (same EUID as ps), or to list all processes when used together with the a option.

-f Do full-format listing. This option can be combined with many other UNIX-style options to add additional columns. It also causes the command arguments to be printed. When used with -L, the NLWP (number of threads) and LWP (thread ID) columns will be added

-u Display user-oriented format

.


tty Man Page | Bash | SS64
10734135_10201856006
scripts : bash - Konsole
ospracticalfile.docx - Libre
Commands.docx - LibreOffi
10:45 AI
Local
scripts : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$ps -ef | grep firefox
bsachde+ 1525 1143 9 Nov23 ? 03:35:26 /usr/lib64/firefox/firefox
bsachde+ 18766 1525 1 01:53 ? 00:05:25 /usr/lib64/firefox/plugin-container /usr/lib64/flash-plugin/libflashplayer.so -greomni /usr/lib64/firefox/omni.ja -appomni /usr/lib64/firefox/browser/omni.ja -appdir /usr/lib64/firefox/browser 1525 true plugin
bsachde+ 24680 7135 0 10:43 pts/2 00:00:00 grep --color=auto firefox
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$ps -aux | grep firefox
bsachde+ 1525 9.3 21.5 2210764 646100 ? Sl Nov23 215:26 /usr/lib64/firefox/firefox
bsachde+ 18766 1.0 2.5 651268 76880 ? Sl 01:53 5:25 /usr/lib64/firefox/plugin-container /usr/lib64/flash-plugin/libflashplayer.so -greomni /usr/lib64/firefox/omni.ja -appomni /usr/lib64/firefox/browser/omni.ja -appdir /usr/lib64/firefox/browser 1525 true plugin
bsachde+ 24682 0.0 0.0 112680 2300 pts/2 S+ 10:44 0:00 grep --color=auto firefox
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$ps -ef | grep cat
bsachde+ 1482 1 0 Nov23 ? 00:00:02 /usr/libexec/kde4/polkit-kde-authentication-agent-1
bsachde+ 24684 1439 0 10:44 pts/0 00:00:00 cat
bsachde+ 24686 7135 0 10:44 pts/2 00:00:00 grep --color=auto cat
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$sudo kill -9 1439
[sudo] password for bsachdeva:
[bsachdeva@localhost Mon Nov 24 06:54:18 IST 2014 scripts]$
scripts : bash

The find command

Search a folder hierarchy for filename(s) that meet a desired criteria: Name, Size, File Type

SYNOPSIS

find [-H] [-L] [-P] [*path...*] [*expression*]

GNU find searches the directory tree rooted at each given file name by evaluating the given *expression* from left to right, according to the rules of precedence, until the outcome is known (the left hand side is false for **AND** operations, true for **OR**), at which point find moves on to the next file name.

DESCRIPTION

-depth
Process each directory's contents before the directory itself.

-d
A synonym for -depth, for compatibility with FreeBSD, NetBSD, MacOS X and OpenBSD.

-maxdepth *levels*
Descend at most levels (a non-negative integer) levels of directories below the command line arguments. '-maxdepth 0' means only apply the tests and actions to the command line arguments.

-mindepth *levels*
Do not apply any tests or actions at levels less than levels (a non-negative integer). '-mindepth 1' means process all files except the command line arguments.

-mount
Don't descend directories on other filesystems. An alternate name for -xdev, for compatibility with some other versions of find.

-noleaf
Do not optimize by assuming that directories contain 2 fewer subdirectories than their hard link count. This option is needed when searching filesystems that do not follow the Unix directory-link convention, such as CD-ROM or MS-DOS filesystems or AFS volume mount points. Each directory on a normal Unix filesystem has at least 2 hard links: its name and its '.' entry. Additionally, its subdirectories (if any) each have a '..' entry linked to that directory. When find is examining a directory, after it has statted 2 fewer subdirectories than the directory's link count, it knows that the rest of the entries in the directory are non-directories ('leaf' files in the directory tree). If only the files' names need to be examined, there is no need to stat them; this gives a significant increase in search speed.

-regextype *type*
Changes the regular expression syntax understood by **-regex** and **-iregex** tests which occur later on the command line. Currently-implemented types are emacs (this is the default), posix-awk, posix-basic, posix-egrep and posix-extended.

-inum *n* File has inode number *n*. It is normally easier to use the **-samefile** test instead.

-nouser No user corresponds to file's numeric user ID.

-nogroup No group corresponds to file's numeric group ID.

-path *pattern* See -wholename. The predicate -path is also supported by HP-UX find.

-perm *mode*
File's permission bits are exactly mode (octal or symbolic). Since an exact match is required, if you want to use this form for symbolic modes, you may have to specify a rather complex mode string. For example '-perm g=w' will only match files which have mode 0020 (that is, ones for which group write permission is the only permission set). It is more likely that you will want to use the '/' or '-' forms, for example '-perm -g=w', which matches any file with group write permission.

-perm -*mode*
All of the permission bits mode are set for the file. Symbolic modes are accepted in this form, and this is usually the way in which would want to use them. You must specify 'u', 'g' or 'o' if you use a symbolic mode.

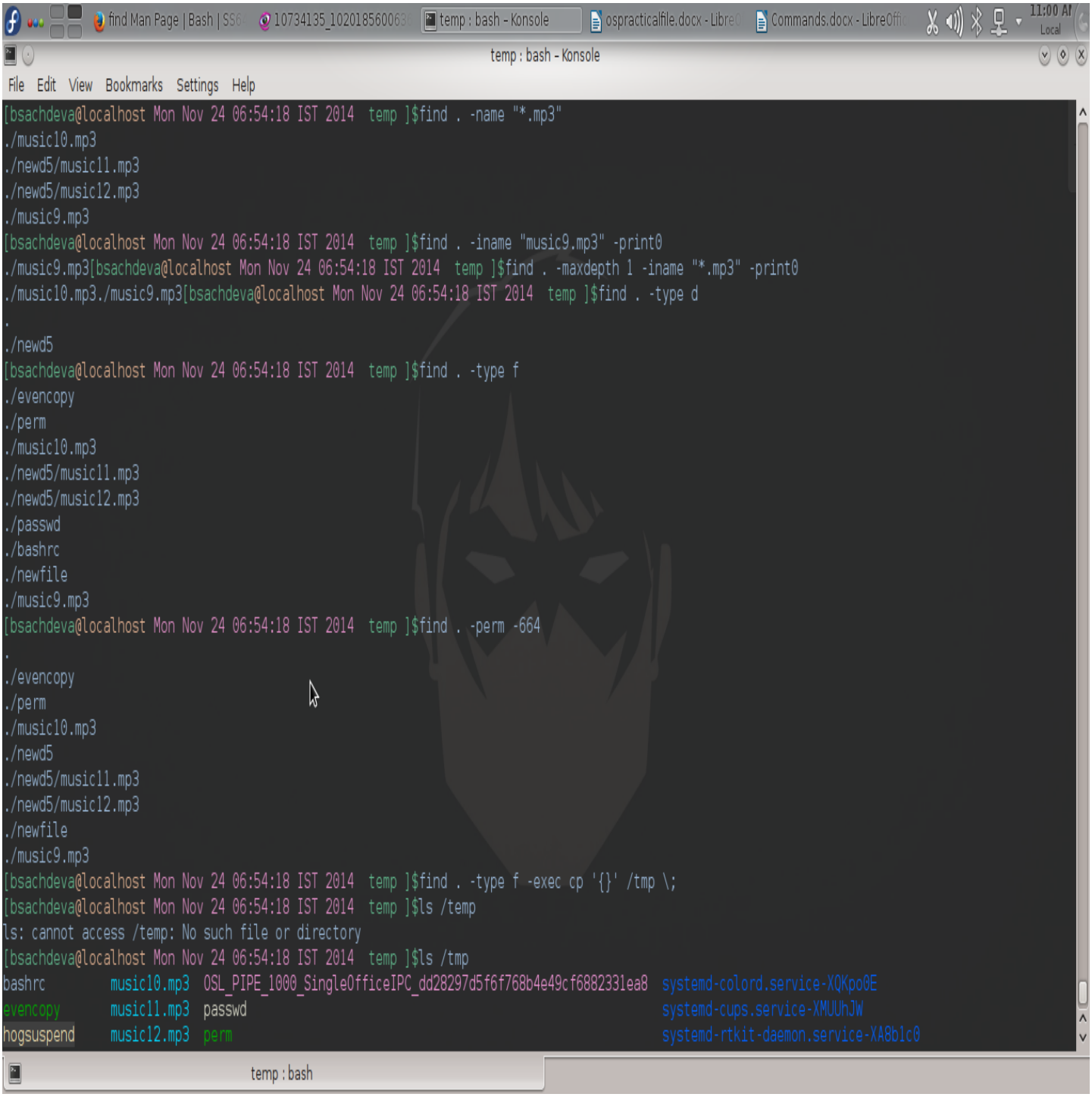
-perm /*mode*
Any of the permission bits mode are set for the file. Symbolic modes are accepted in this form. You must specify 'u', 'g' or 'o' if you use a symbolic mode. If no permission bits in mode are set, this test currently matches no files. However, it will soon be changed to match any file (the idea is to be more consistent with the behaviour of **perm -000**).

-perm +*mode*
Deprecated, old way of searching for files with any of the permission bits in mode set. You should use **-perm** /mode instead. Trying to use the '+' syntax with symbolic modes will yield surprising results. For example, '+u+x' is a valid symbolic mode (equivalent to +u,+x, i.e. 0111) and will therefore not be evaluated as **-perm** +mode but instead as the exact mode specifier **-perm** mode and so it matches files with exact permissions 0111 instead of files with any execute bit set. If you found this paragraph confusing, you're not alone - just use -perm /mode. This form of the **-perm** test is deprecated because the POSIX specification requires the interpretation of a leading '+' as being part of a symbolic mode, and so we switched to using '/' instead.

-exec *command* ;
Execute command; true if 0 status is returned. All following arguments to **find** are taken to be arguments to the command until an argument consisting of ';' is encountered. The string '{}' is replaced by the current file name being processed everywhere it occurs in the arguments to the command, not just in arguments where it is alone, as in some versions of **find**. Both of these constructions might need to be escaped (with a '\') or quoted to protect them from expansion by the shell. See the EXAMPLES section for examples of the use of the '-exec' option. The specified command is run once for each matched file. The command is executed in the starting directory. There are unavoidable security problems surrounding use of the -exec option; you should use the -execdir option instead.

-exec *command* {} +

This variant of the -exec option runs the specified command on the selected files, but the command line is built by appending each selected file name at the end; the total number of invocations of the command will be much less than the number of matched files. The command line is built in much the same way that xargs builds its command lines. Only one instance of '{}' is allowed within the command. The command is executed in the starting directory.



The useradd Command

The useradd command is used on Linux and other Unix-like operating systems both to add new users (also referred to as accounts), inclusive of their user names, passwords and other data, and to update default new user information. The adduser command is identical to useradd, because it is merely a

symbolic link to it.

SYNOPSIS

useradd [option(s)] username

By default, useradd can only be used by the root (i.e., omnipotent administrative) account. On home computers and other computers over which the user has complete control, the root account can be accessed by logging in as root or, preferably, by using the su (i.e., substitute user) command.

The useradd binary (i.e., the ready-to-run program file) is typically located in the /usr/sbin directory, which contains non-vital system utilities that are used after booting (i.e., starting the system). If this directory is not in the root user's PATH (i.e., the directories in which the system searches for commands issued by the root user), an error message such as bash: useradd: command not found will be displayed. In such case, the command should be typed using its full path (i.e., the full hierarchy of directories from it to the root directory), which would be /usr/sbin/useradd.

To add a new user to the system, all that is necessary is to follow the useradd or adduser command by that user's user name. This name is the login name, that is, the name that the user uses when logging into the system. Only one name can be added with each use of the command, and the name must be unique (i.e., different from every other user name already on the system). Thus, for example, to add a new user with a user name joseph to the system, all that is necessary is to type the following and press the Enter key:

/usr/sbin/useradd joseph

Adding a new user automatically adds an entry to the /etc/passwd file, which is used to store data about users. Each entry in this file consists of a single line containing a set of seven colon-separated fields, for example,

joseph:x:504:504:Joseph:/home/joseph:/bin/bash

The fields are the user's login name, password (or just the letter x or an asterisk in the likely event that shadow passwords are used), UID (user identification number), GID (group identification number), comment (optional), user's home directory and default shell (usually bash on Linux). On a system with shadow passwords enabled, the actual passwords are encrypted and stored in a separate file which is only accessible to the root account.

/etc/passwd can be edited by the root user with the use of any text editor, such as vi or gedit, and it can be viewed by any user by using one of these editors or by employing a command such as cat, i.e.,

cat /etc/passwd

adduser's flexibility is enhanced by the availability of a number of options. Among the more commonly used is -c, which allows a comment, such as the user's real name and phone number, to be added to /etc/passwd. The comment can be written as a single string (i.e., a sequence of characters without any intervening spaces), or it can be multiple strings enclosed in quotes. For example, the following would add a user named joe and would insert that user's full name, Joe Smith, into the comment field:

```
useradd -c "Joe Smith" joe
```

By default the home directory of each user (i.e., the directory that contains a user's personal files and that a user is first in after logging in) will reside in the /home directory and will have the same name as the user. Thus, for example, the default home directory for the new user joe would be /home/joe. However, this behavior can be overriden by using the -d option followed by the full path for the desired new home directory (which need not be in /home). For example, the following would create a new user named jim with a home directory /home/james:

```
useradd -d /home/james jim
```

The -e option is used to set the date, in YYYY-MM-DD format, on which the new account will automatically expire. This is useful for creating temporary accounts and for dealing with the tendency of busy system administrators to forget to delete them, which can become a security risk.

The -f option is used to specify the number of days after a password expires until the account is permanently disabled. A value of 0 disables the account as soon as the password has expired. The default value is -1, which disables this feature.

The name or number of the new user's initial login group can be specified by placing it after the -g option. The group name must already exist, and a group number must refer to an already existing group. The default group number is 1 or whatever is specified in the file /etc/default/useradd.

The -G option is used to provide a list of additional groups into which the user is also included. Each group name or number is separated from the next by a comma, with no intervening spaces. The groups are subject to the same restrictions as the group specified with the -g option.

The -m option, which does not accept any arguments (i.e., input data), creates the user's home directory; however, it is not necessary to use it on Red Hat and other systems on which such directory is created automatically. The -k option is used together with -m to specify an alternative to the default /etc/skel directory as the source of the initial content (i.e., directories and files) for the new user's home directory. For example, the following would add the contents of a directory named /dir_1 to /home/jane, which would be the default home directory of a new user jane:

```
useradd -m -k /dir_1 jane
```

The -M option is used to create a new user without creating any home directory for that user. When such a user logs into a system that has just booted up, its login directory will likely be the root directory; when such a user logs into a system using the su command, its login directory will be the current directory of the previous user.

On Red Hat systems a new group having the same name as the user being added to the system will be created by default. The -n option is used to prevent such group from being created. The -o option allows creation of a user with a non-unique UID.

The -p option is used to specify an encrypted password that has been created by the crypt command. It is not used to specify the actual password, as this could be a security risk (because such password

would be retained in the history file).

The -r option is used to create accounts for administrative use that have some, but not all, root privileges. Such accounts have a UID lower than the value of UID_MIN defined in /etc/login.defs (typically 500 and above for ordinary users) and have a password that does not expire. By default, no home directories are created.

New users are automatically assigned the default login shell, which is bash on Linux unless it has been changed systemwide. However, any new user can be assigned any other shell as their default by using the -s option followed by the name of the desired shell.

The -D option is used to view and change the default values for adding users. The following will display the default settings, which include the initial group, location of the home directory, when the password becomes inactive, when the account expires, the default shell, and name of the directory that contains the directories and/or files that are added automatically to the new home directory:

useradd -D

To change any of the default settings for new users, -D is followed by -g, -b, -s, -f and/or -e, which are, in turn, followed by an appropriate argument. These secondary options are the same as their counterparts described above, with the exception of -b, which stands for base and is followed by the full path of the directory in which new home directories are to be created. Thus, for example, to cause the home directory for new users to be located in a directory called /home/users, the following would be used:

useradd -Db /home/users

Any directory that is specified as the container for new users' home directories need not exist at the time it is specified with -Db. But it must exist when new users are subsequently created. This can, of course, be easily accomplished using the mkdir command, which for the above example would be

mkdir /home/users

As an alternative to using the useradd command, users can also be added and their data changed by directly editing /etc/passwd. However, this is more risky because of the possibility of accidental damage to the file. If a critical system file such as /etc/passwd is to be edited directly, it is, of course, wise to first make a backup copy of it.

After adding a new user, the system administrator uses the passwd command to assign that user a password. User accounts can be removed, preferably, with the userdel command but also by deleting the appropriate line from /etc/passwd.

DESCRIPTION

-b, --base-dir BASE_DIR

The default base directory for the system if -d HOME_DIR is not specified. BASE_DIR is concatenated with the account name to define

the home directory. If the -m option is not used, BASE_DIR must exist.

If this option is not specified, useradd will use the base directory specified by the HOME variable in /etc/default/useradd, or /home

by default.

-c, --comment COMMENT

Any text string. It is generally a short description of the login, and is currently used as the field for the user's full name.

-d, --home HOME_DIR

The new user will be created using HOME_DIR as the value for the user's login directory. The default is to append the LOGIN name to BASE_DIR and use that as the login directory name. The directory HOME_DIR does not have to exist but will not be created if it is

missing.

-D, --defaults

See below, the subsection "Changing the default values".

-e, --expiredate EXPIRE_DATE

The date on which the user account will be disabled. The date is specified in the format YYYY-MM-DD. If not specified, useradd will use the default expiry date specified by the EXPIRE variable in /etc/default/useradd, or an empty string (no expiry) by default.

-g, --gid GROUP

The group name or number of the user's initial login group. The group name must exist. A group number must refer to an already existing group.

If not specified, the behavior of useradd will depend on the USERGROUPS_ENAB variable in /etc/login.defs. If this variable is set to yes (or -U/--user-group is specified on the command line), a group will be created for the user, with the same name as her loginname.

If the variable is set to no (or -N/--no-user-group is specified on the command line), useradd will set the primary group of the new user to the value specified by the GROUP variable in /etc/default/useradd, or 100 by default.

-G, --groups GROUP1[,GROUP2,...[,GROUPN]]]

A list of supplementary groups which the user is also a member of. Each group is separated from the next by a comma, with no intervening whitespace. The groups are subject to the same restrictions as the group given with the -g option. The default is for the user to belong only to the initial group.

-k, --skel SKEL_DIR

The skeleton directory, which contains files and directories to be copied in the user's home

directory, when the home directory is created by useradd.

This option is only valid if the -m (or --create-home) option is specified.

If this option is not set, the skeleton directory is defined by the SKEL variable in /etc/default/useradd or, by default, /etc/skel.

-m, --create-home

Create the user's home directory if it does not exist. The files and directories contained in the skeleton directory (which can be defined with the -k option) will be copied to the home directory.

By default, if this option is not specified and CREATE_HOME is not enabled, no home directories are created.

-o, --non-unique

Allow the creation of a user account with a duplicate (non-unique) UID.

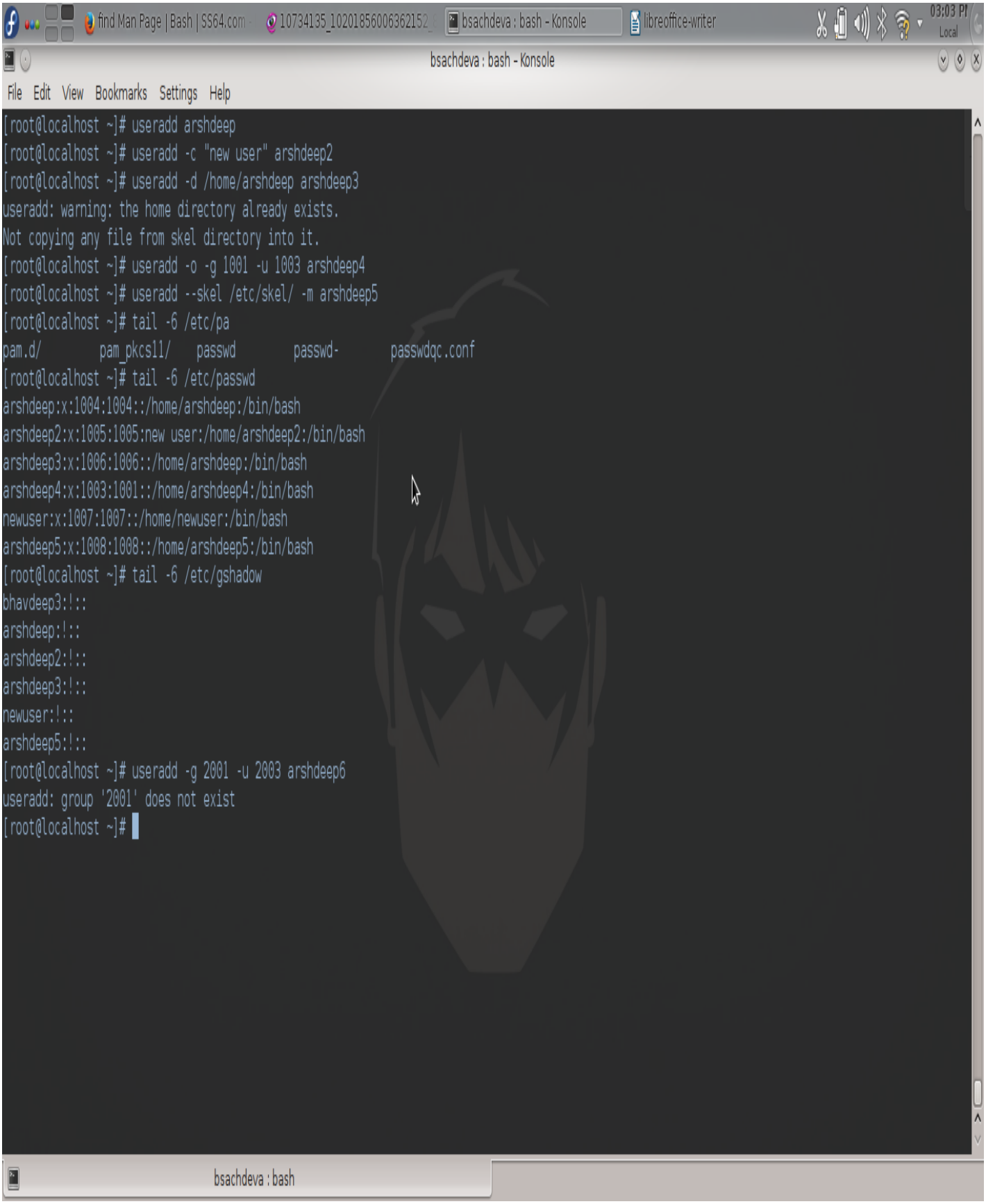
This option is only valid in combination with the -u option.

-s, --shell SHELL

The name of the user's login shell. The default is to leave this field blank, which causes the system to select the default login shell specified by the SHELL variable in /etc/default/useradd, or an empty string by default.

-u, --uid UID

The numerical value of the user's ID. This value must be unique, unless the -o option is used. The value must be non-negative. The default is to use the smallest ID value greater than or equal to UID_MIN and greater than every other user.


find Man Page | Bash | SS64.com
bsachdeva : bash - Konsole
libreoffice-writer
03:03 PM
Local
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[root@localhost ~]# useradd arshdeep
[root@localhost ~]# useradd -c "new user" arshdeep2
[root@localhost ~]# useradd -d /home/arshdeep arshdeep3
useradd: warning: the home directory already exists.
Not copying any file from skel directory into it.
[root@localhost ~]# useradd -o -g 1001 -u 1003 arshdeep4
[root@localhost ~]# useradd --skel /etc/skel/ -m arshdeep5
[root@localhost ~]# tail -6 /etc/pa
pam.d/ pam_pkcs11/ passwd passwd- passwdqc.conf
[root@localhost ~]# tail -6 /etc/passwd
arshdeep:x:1004:1004::/home/arshdeep:/bin/bash
arshdeep2:x:1005:1005:new user:/home/arshdeep2:/bin/bash
arshdeep3:x:1006:1006::/home/arshdeep:/bin/bash
arshdeep4:x:1003:1001::/home/arshdeep4:/bin/bash
newuser:x:1007:1007::/home/newuser:/bin/bash
arshdeep5:x:1008:1008::/home/arshdeep5:/bin/bash
[root@localhost ~]# tail -6 /etc/gshadow
bhavdeep3:!::
arshdeep:!::
arshdeep2:!::
arshdeep3:!::
newuser:!::
arshdeep5:!::
[root@localhost ~]# useradd -g 2001 -u 2003 arshdeep6
useradd: group '2001' does not exist
[root@localhost ~]#
bsachdeva : bash

The shutdown command

Shutdown or restart linux

SYNOPSIS

shutdown [*options*] *when* [*message*]

DESCRIPTION

-c Cancel a shutdown that is in progress.

-f Reboot fast, by suppressing the normal call to fsck when rebooting.

-h Halt the system when shutdown is complete.

-k Print the warning message, but suppress actual shutdown.

-n Perform shutdown without a call to ini.

-r Reboot the system when shutdown is complete.

-t *sec*

Ensure a sec-second delay between killing processes and changing the run level.

Examples

```
 Shutdown immediately:
   shutdown -h now

 Reboot immediately:
   shutdown -r now

 Shutdown at 8 pm:
   shutdown -h 20:00

 Shutdown in 10 minutes:
   shutdown -h +10
```

The mail Command

It is used to send and receive Internet mail. It can be also used for mail on intranet it is command-line based mail service. It is an intelligent mail processing system, which has a command syntax reminiscent of ed(1) with lines replaced by messages. It is based on Berkeley Mail 8.1, is intended to provide the functionality of the POSIX mailx command, and offers extensions for MIME, IMAP,POP3, SMTP, and S/MIME. Mailx provides enhanced features for interactive use, such as caching and disconnected operation for IMAP, message threading, scoring, and filtering. It is also usable as a mail batch language, both for sending and receiving mail.

SYNOPSIS

mail [Option] to-addr ...

DESCRIPTION

-e Just check if mail is present in the system mailbox. If yes, return an exit status of zero, else, a non-zero value.

-H Print header summaries for all messages and exit.

-h hops

Invoke send mail with the specified hop count. This option has no effect when SMTP is used for sending mail.

-q file

Start the message with the contents of the specified file. May be given in send mode only.

-r address

Sets the From address. Overrides any from variable specified in environment or startup files. Tilde escapes are disabled. The -r address options are passed to the mail transfer agent unless SMTP is used. This option exists for compatibility only; it is recommended to set the from variable directly instead.

-s subject

Specify subject on command line (only the first argument after the -s flag is used as a subject; be careful to quote subjects containing spaces).

The split Commands

Split a file into fixed-size pieces, creates output files containing consecutive sections of *INPUT* (standard input if none is given or INPUT is `-')

SYNOPSIS
split [*options*] [*INPUT* [*PREFIX*]]
DESCRIPTION
-*LINES*
-l *LINES*
--lines=*LINES*
Put *LINES* lines of *INPUT* into each output file.
-b *BYTES*
--bytes=*BYTES*
Put the first *BYTES* bytes of *INPUT* into each output file.
Appending `b' multiplies *BYTES* by 512, `k' by 1024, and `m' by 1048576.
-C *BYTES*
--line-bytes=*BYTES*
Put into each output file as many complete lines of INPUT as possible without exceeding BYTES bytes. For lines longer than BYTES bytes, put BYTES bytes into each output file until less than BYTES bytes of the line are left, then continue normally. BYTES has the same format as for the `--bytes' option.
--verbose
Write a diagnostic to standard error just before each output file is opened.

By default, `split' puts 1000 lines of *INPUT* (or whatever is left over for the last section), into each output file.
The output files' names consist of *PREFIX* (`x' by default) followed by a group of letters `aa', `ab', and so on, such that concatenating the output files in sorted order by file name produces the original input file.
If more than 676 output files are required, `split' uses `zaa', `zab', etc.

Examples

Split up the file demo.zip into multiple 100 MB files:

$ split -b 100m demo.zip

The output files will be named with 3 letters starting xaa, xab, … to reassemble them, cat the files in a lphabetical order:

$ cat `ls x*` > demo2.zip

The userdel Command

Delete a user account and related files.

SYNOPSIS

userdel [*options*] *LOGIN*

userdel modifies the system account files, deleting all entries that refer to the user name *LOGIN*. The named user must exist.

DESCRIPTION

-f, --force

This option forces the removal of the user account, even if the user is still logged in. It also forces userdel to remove the user's home directory and mail spool, even if another user uses the same home directory or if the mail spool is not owned by the specified user.
If USERGROUPS_ENAB is defined to yes in /etc/login.defs and if a group exists with the same name as the deleted user, then this group will be removed, even if it is still the primary group of another user.

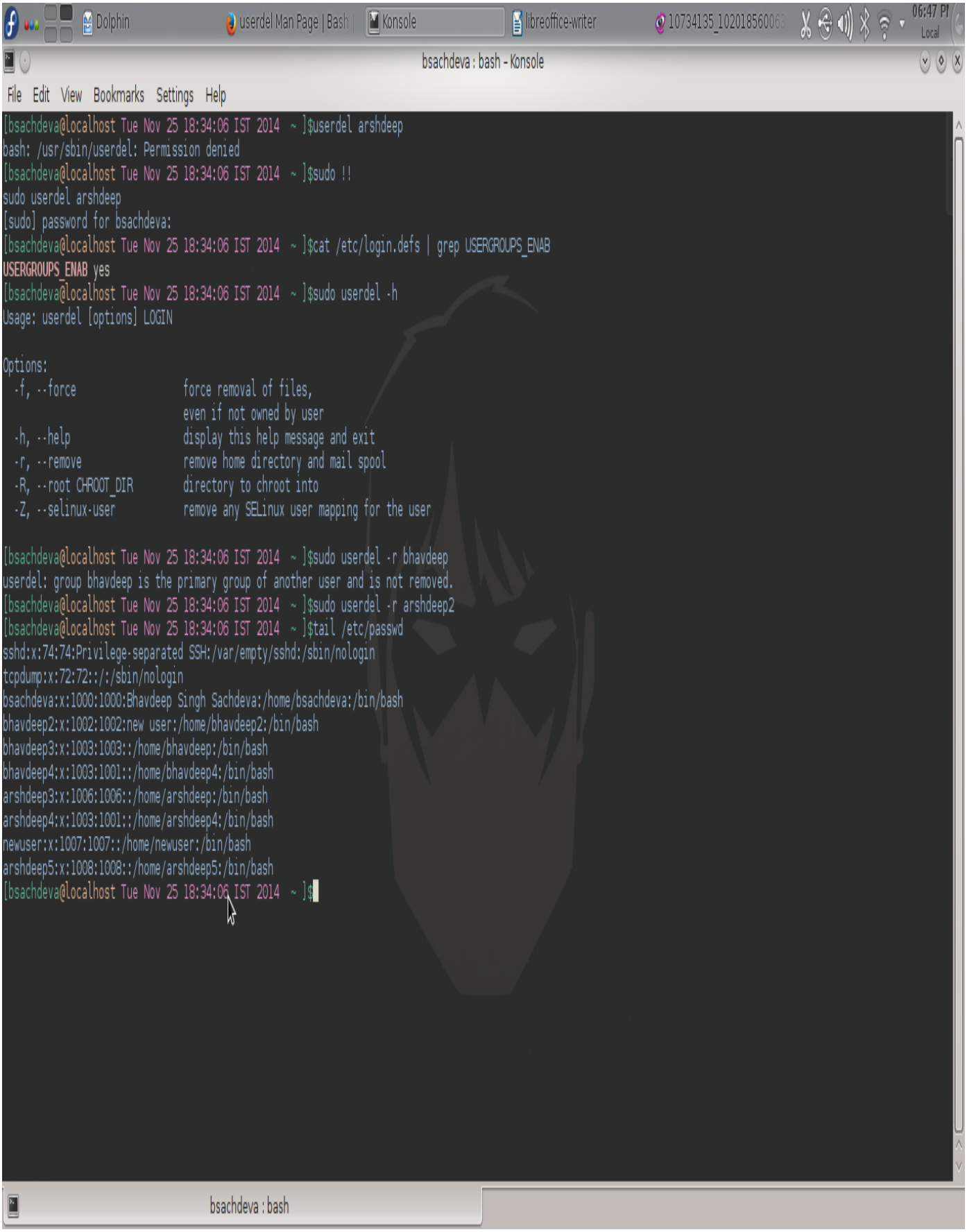
Note: This option is dangerous and may leave your system in an inconsistent state.

-h, --help

Display help message and exit.

-r, --remove

Files in the user's home directory will be removed along with the home directory itself and the user's mail spool.
Files located in other file systems will have to be searched for and deleted manually.


Dolphin
userdel Man Page | Bash
Konsole
libreoffice-writer
06:47 PM
Local
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$userdel arshdeep
bash: /usr/sbin/userdel: Permission denied
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$sudo !!
sudo userdel arshdeep
[sudo] password for bsachdeva:
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$cat /etc/login.defs | grep USERGROUPS_ENAB
USERGROUPS_ENAB yes
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$sudo userdel -h
Usage: userdel [options] LOGIN
Options:
-f, --force force removal of files,
even if not owned by user
-h, --help display this help message and exit
-r, --remove remove home directory and mail spool
-R, --root CHROOT_DIR directory to chroot into
-Z, --selinux-user remove any SELinux user mapping for the user
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$sudo userdel -r bhavdeep
userdel: group bhavdeep is the primary group of another user and is not removed.
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$sudo userdel -r arshdeep2
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$tail /etc/passwd
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
bsachdeva:x:1000:1000:Bhavdeep Singh Sachdeva:/home/bsachdeva:/bin/bash
bhavdeep2:x:1002:1002:new user:/home/bhavdeep2:/bin/bash
bhavdeep3:x:1003:1003::/home/bhavdeep:/bin/bash
bhavdeep4:x:1003:1001::/home/bhavdeep4:/bin/bash
arshdeep3:x:1006:1006::/home/arshdeep:/bin/bash
arshdeep4:x:1003:1001::/home/arshdeep4:/bin/bash
newuser:x:1007:1007::/home/newuser:/bin/bash
arshdeep5:x:1008:1008::/home/arshdeep5:/bin/bash
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$
bsachdeva : bash

The nice command

Run a command with modified scheduling priority, print or modify the scheduling priority of a job.

SYNOPSIS

nice [*Option*]... [*Command* [*Arg*]...]

DESCRIPTION

-n *MyADJUSTMENT*
-MyADJUSTMENT
--adjustment=*MyADJUSTMENT*

Priority can be adjusted by `nice' over the range
of -20 (the highest priority)
to 19 (the lowest)

If no arguments are given, `nice' prints the current scheduling priority, which it inherited.

Otherwise, `nice' runs the given *Command* with its scheduling priority adjusted.

If no option for *MyADJUSTMENT* is given, the priority of the command is incremented by 10. You must have appropriate privileges to specify a negative adjustment.

Because most shells have a built-in command by the same name, using the unadorned command name in a script or interactively may get you different functionality than that described here.

Examples

Run an echo command with low priority:
$ sudo nice -n 10 echo Hello

Run an echo command with high priority:
$ sudo nice -n -10 echo Hello

The crontab command

It maintains crontab files for individual users

Crontab is the program used to install, remove or list the tables used to serve the cron daemon. Each user can have their own crontab, and though these are files in /var/spool/, they are not intended to be edited directly. For SELinux in MLS mode, you can define more crontabs for each range.

In this version of Cron it is possible to use a network-mounted shared /var/spool/cron across a cluster of hosts and specify that only one of the hosts should run the crontab jobs in the particular directory at any one time. You may also use crontab(1) from any of these hosts to edit the same shared set of crontab files, and to set and query which host should run the crontab jobs.

Running cron jobs can be allowed or disallowed for different users. For this purpose, use the cron.allow and cron.deny files. If the cron.allow file exists, a user must be listed in it to be allowed to use cron If the cron.allow file does not exist but the cron.deny file does exist, then a user must not be listed in the cron.deny file in order to use cron. If neither of these files exists, only the super user is allowed to use cron. Another way to restrict access to cron is to use PAM authentication in /etc/security/access.conf to set up users, which are allowed or disallowed to use crontab or modify system cron jobs in the /etc/cron.d/ directory

DESCRIPTION

-u Appends the name of the user whose crontab is to be modified. If this option is not used, crontab examines "your" crontab, i.e., the crontab of the person executing the command. Note that su(8) may confuse crontab, thus, when executing commands under su(8) you should always use the -u option.If no crontab exists for a particular user, it is created for him the first time the crontab -u command is used under his username.

-l Displays the current crontab on standard output.

-r Removes the current crontab.

-e Edits the current crontab using the editor specified by the VISUAL or EDITOR environment variables. After you exit from the editor, the modified crontab will be installed automatically.

-i This option modifies the -r option to prompt the user for a 'y/Y' response before actually removing the crontab.

-s Appends the current SELinux security context string as an MLS_LEVEL setting to

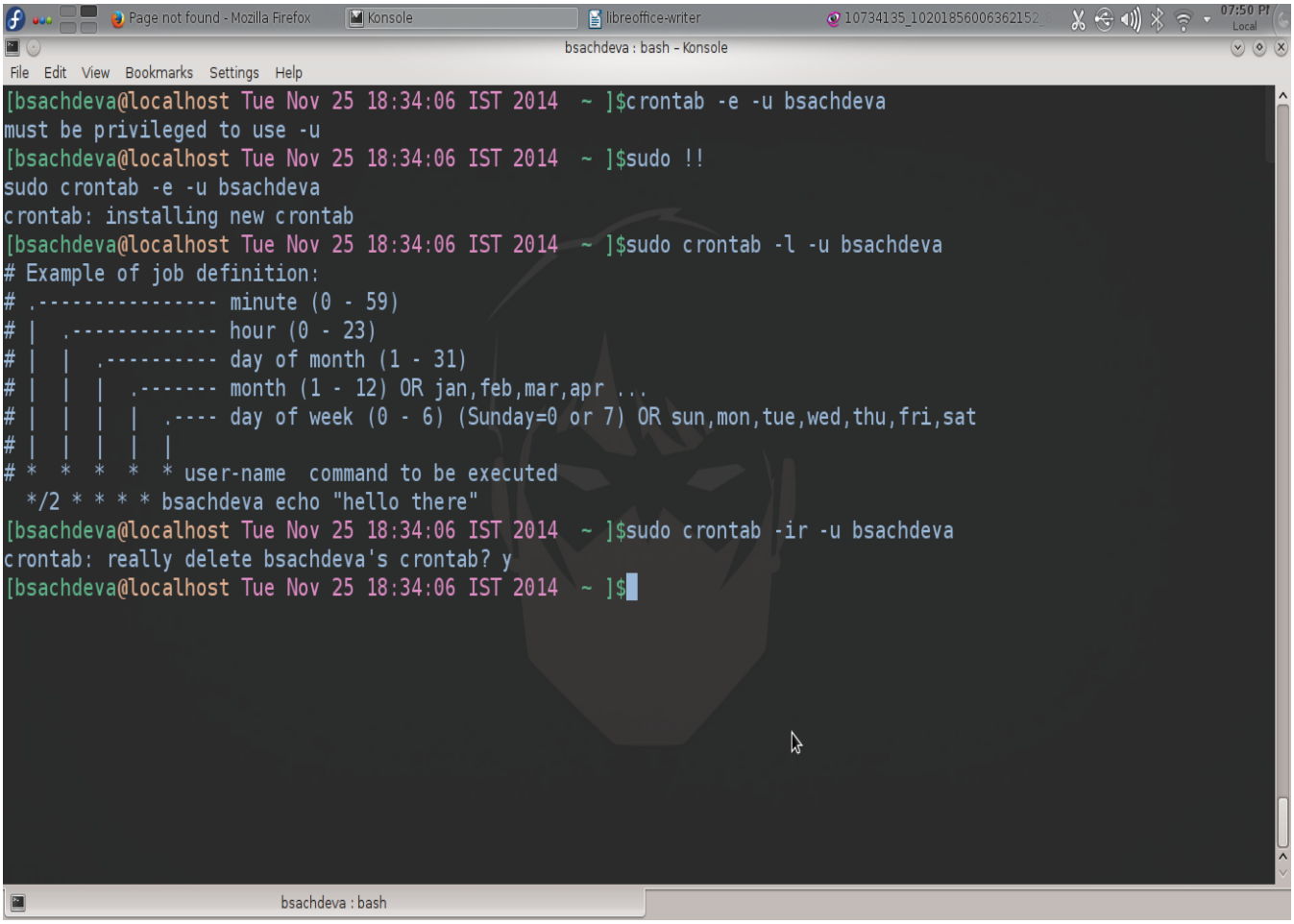
the crontab file before editing / replacement occurs - see the documentation of MLS_LEVEL in crontab(5).

-n This option is relevant only if cron(8) was started with the -c option, to enable clustering support. It is used to set the host in the cluster which should run the jobs specified in the crontab files in the /var/spool/cron directory. If a hostname is supplied, the host whose hostname

returned by gethostname(2) matches the supplied hostname, will be selected to run the selected cron jobs subsequently. If there is no host in the

cluster matching the supplied hostname, or you explicitly specify an empty hostname, then the selected jobs will not be run at all. If the hostname is omitted, the name of the local host returned by gethostname(2) is used. Using this option has no effect on the /etc/crontab file and the files in the /etc/cron.d directory, which are always run, and considered host-specific..

-c This option is only relevant if cron(8) was started with the -c option, to enable clustering support. It is used to query which host in the cluster is currently set to run the jobs specified in the crontab files in the directory /var/spool/cron , as set using the -n option.


Page not found - Mozilla Firefox
Konsole
libreoffice-writer
07:50 PM
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$crontab -e -u bsachdeva
must be privileged to use -u
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$sudo !!
sudo crontab -e -u bsachdeva
crontab: installing new crontab
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$sudo crontab -l -u bsachdeva
Example of job definition:
.---------------- minute (0 - 59)
| .------------- hour (0 - 23)
| | .---------- day of month (1 - 31)
| | | .------- month (1 - 12) OR jan,feb,mar,apr ...
| | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
| | | | |
* * * * * user-name command to be executed
*/2 * * * * bsachdeva echo "hello there"
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$sudo crontab -ir -u bsachdeva
crontab: really delete bsachdeva's crontab? y
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$
bsachdeva : bash

The kill Command

The kill command is used on Linux and other Unix-like operating systems to terminate processes without having to log out or reboot (i.e., restart) the computer. Thus, it is particularly important to the stability of such systems.

A process, also referred to as a task, is an executing (i.e., running) instance of a program. Each process is automatically assigned a unique process identification number (PID) when it is created for use by the system to reference the process.

SYNOPSIS

kill [signal or option] PID(s)

The only argument (i.e., input) that is required is a PID, and as many PIDs as desired can be used in a single command. Typically no signal or option is used.

Thus, if it is desired to terminate a process with a PID of 485, the following will usually be sufficient:

kill 485

The kill command has a misleading name because it does not actually kill processes. Rather, it sends signals to them. Each process is supplied with a set of standard signal handlers by the operating system in order to deal with incoming signals. When no signal is explicitly included in the command, signal 15, named SIGTERM, is sent by default. If this fails, the stronger signal 9, called SIGKILL, should be used. For example, the following command would nearly guarantee that process 485 would be killed:

kill -9 485

The only situation in which signal 9 will fail is if the process is in the midst of making a system call, which is a request to the kernel (i.e., the core of the operating system) for some action such as process creation. In this situation, the process will die once it has returned from the system call.

There are more than 60 signals that can be used with kill, but, fortunately, most users will only need to be aware of signal 9. The full list is contained in the file /usr/include/linux/signal.h and can be viewed by using kill with its -l option, i.e.,

kill -l

Obvious signs of misbehaving processes are programs that crash (i.e., appear to freeze or otherwise stop operating as expected) or that cannot be shut down normally (e.g., by clicking on a button or using using a menu command). The first step in such situation is to obtain the PID(s) of the offending process(es). This can often be accomplished with the help of the ps command, usually with its -a, -u and -x options (which tell it to list all processes and provide detailed information about them), i.e.,

ps -aux | less

As the list of processes can be quite long, the output of ps -aux can be piped (i.e., transferred) to the

less command, which lets it be viewed one screenful at a time. The list can be advanced one screen forward by pressing the SPACE bar and one screen backward by pressing the b key.

Often it is obvious which is the offending process. However, sometimes it is not, particularly when the running of a program involves multiple processes, and thus it might be necessary to terminate several processes in order to close the offending program.

The pstree command can also be a useful tool for finding offending processes. This is because it displays the names of all processes on the system in the form of a tree diagram, thereby showing all of their parent/child relationships. When used with its -p option, pstree also shows the PIDs of the processes, i.e.,

```
pstree -p | less
```

pstree can simplify terminating a series of related processes (i.e., a process and all of its descendants) because it makes it immediately clear which process is the parent; all that is necessary is to kill the parent in order to also terminate all of its descendant processes. That is, it is not necessary to manually search through a list of processes to find and individually terminate each one as would be necessary using ps.

Because Unix-like operating systems and many of their application programs are inherently very robust (i.e., stable and resistant to crashing), it is not necessary to use the kill command as often as it is to terminate programs or reboot on some other operating systems. However, it does occasionally come in quite handy.

```
# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name  command to be executed
```

Despite its usefulness, however, there are situations in which the kill command even with its strongest signal is insufficient to terminate a process and rescue a frozen system or in which it is difficult to determine what is the offending process (or processes). In such case it is often easier to just reboot the system.

DESCRIPTION

The command kill sends the specified signal to the specified process or process group. If no signal is specified, the TERM signal is sent.

The TERM signal will kill processes which do not catch this signal. For other processes, it may be necessary to use the KILL (9) signal, since this signal cannot be caught.

Most modern shells have a builtin kill function, with a usage rather similar to that of the command described here. The '-a' and '-p' options, and the possibility to specify processes by command name are a local extension.

If sig is 0, then no signal is sent, but error checking is still performed.

n where n is larger than 0. The process with pid n will be signaled.

0 All processes in the current process group are signaled.

-1 All processes with pid larger than 1 will be signaled.

-n where n is larger than 1. All processes in process group n are signaled. When an argument of the form '-n' is given, and it is meant to denote a process group, either the signal must be specified first, or the argument must be preceded by a '--' option, otherwise it will be taken as the signal to send.

-s, --signal signal

Specify the signal to send. The signal may be given as a signal name or number.

-l, --list [signal]

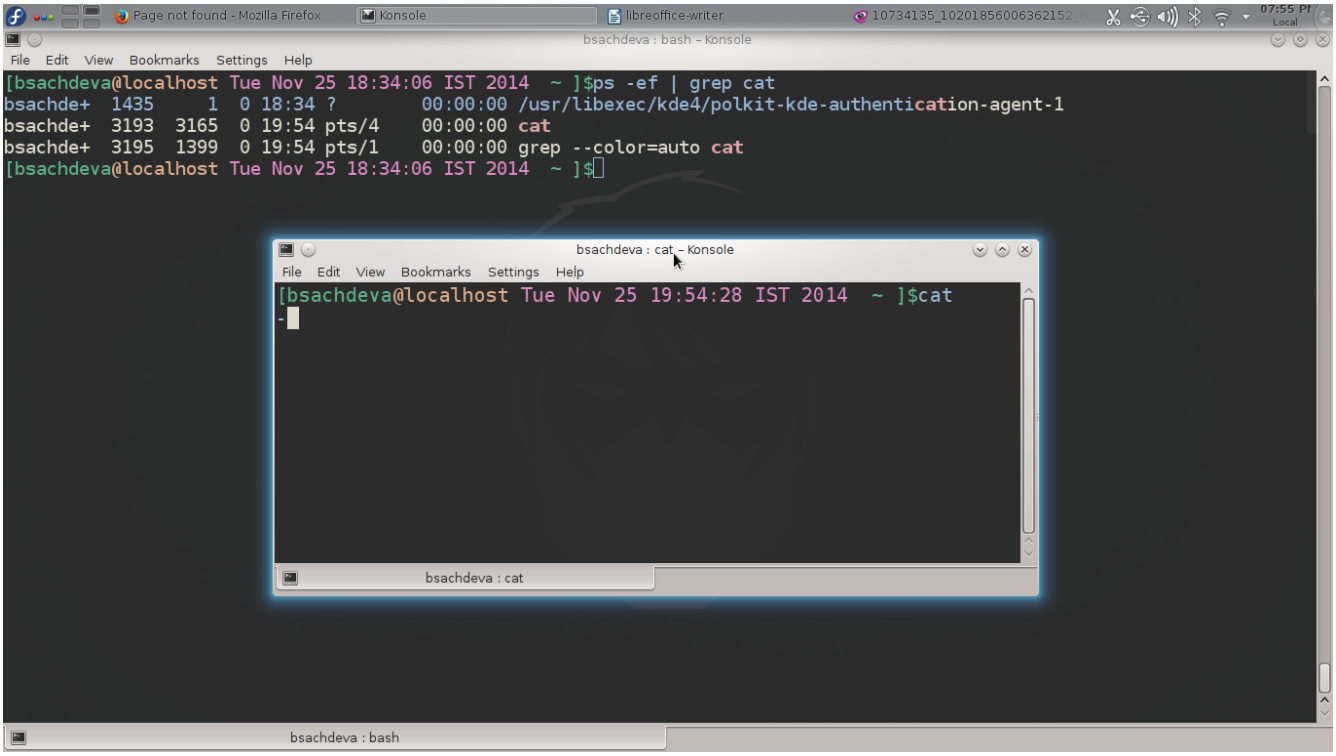
Print a list of signal names, or convert signal given as argument to a name. The signals are found in /usr/include/linux/signal.h

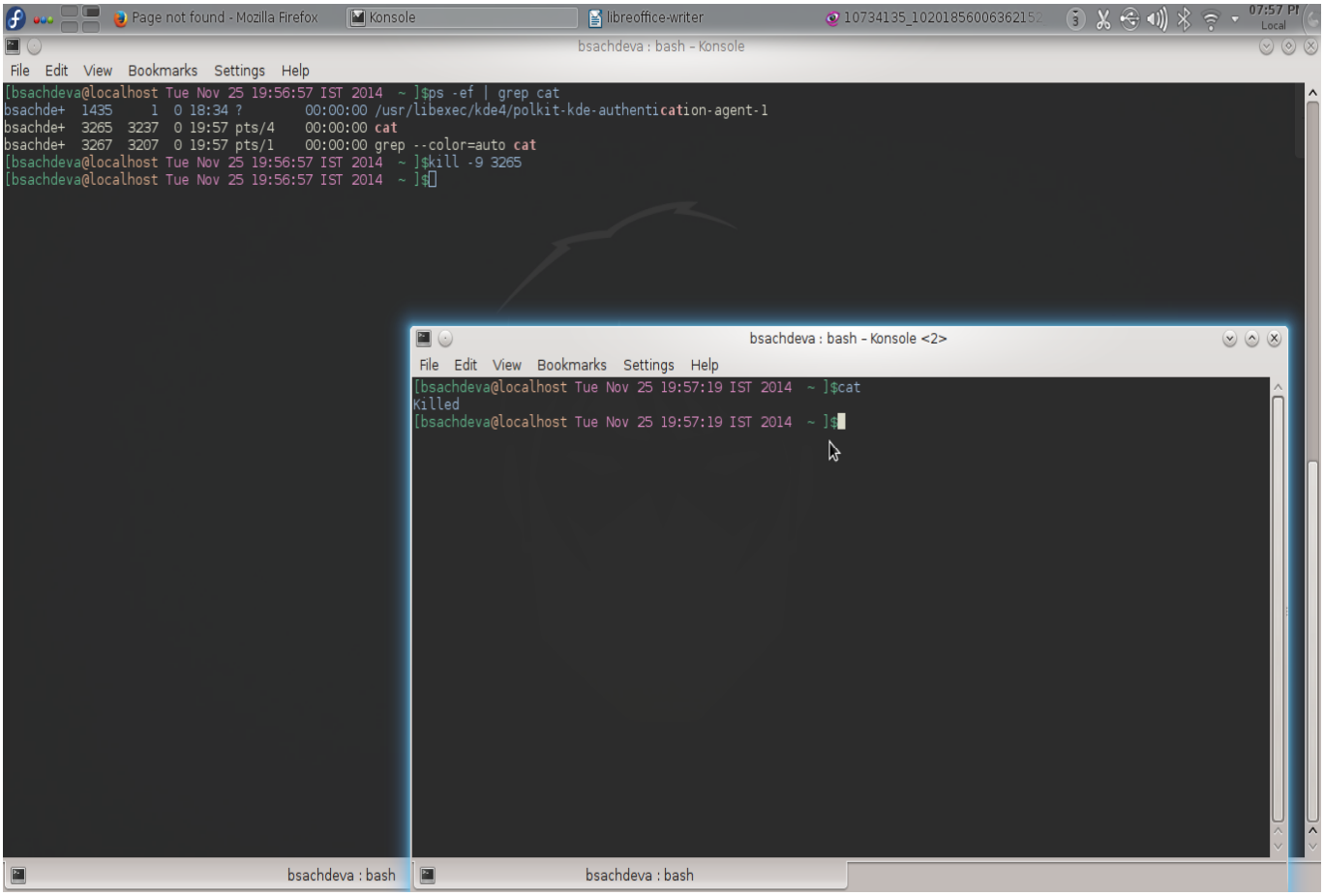
-L, --table

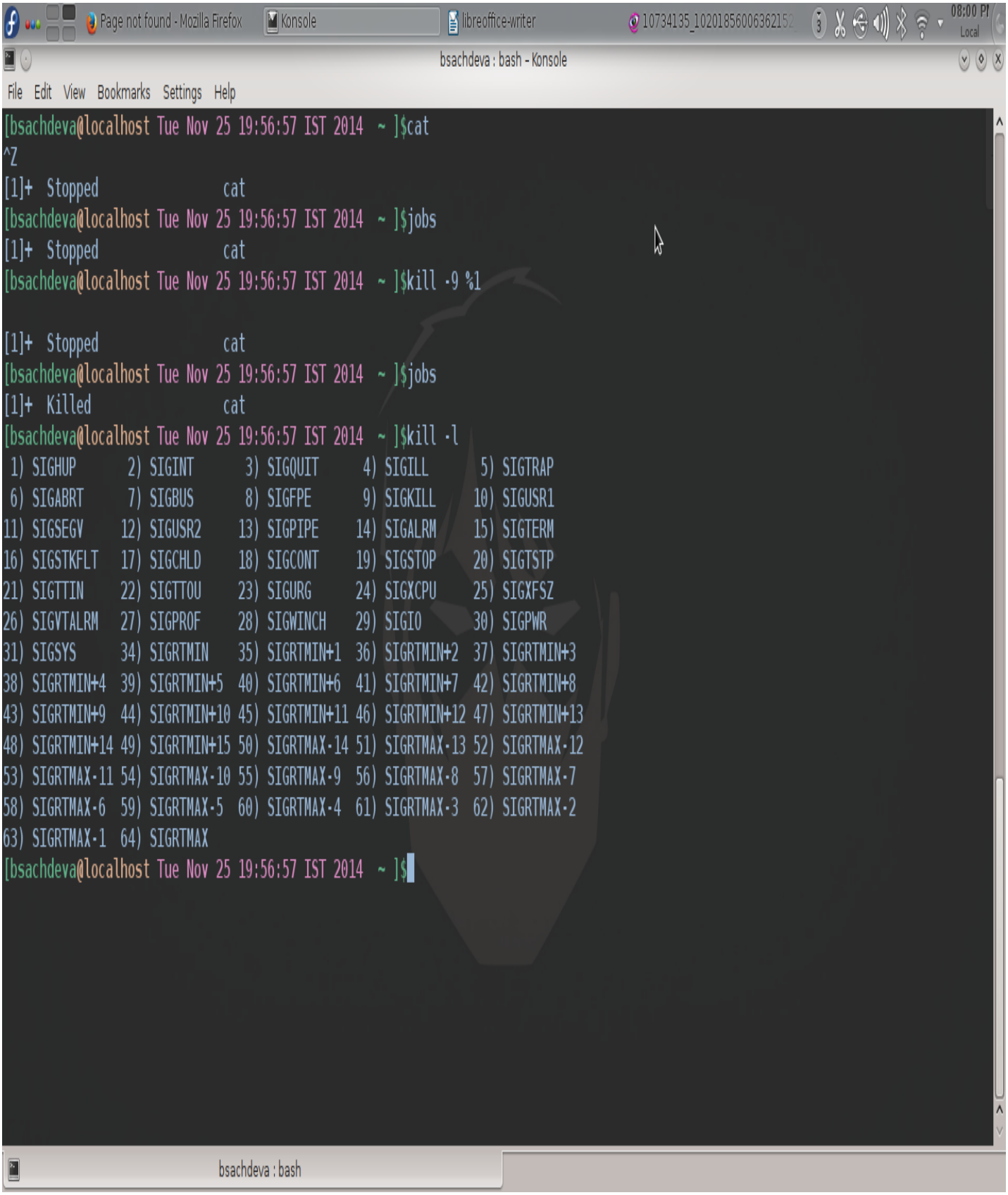
Similar to -l, but will print signal names and their corresponding numbers.

-a, --all

Do not restrict the commandname-to-pid conversion to processes with the same uid as the present process.


Page not found - Mozilla Firefox
Konsole
libreoffice-writer
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$ps -ef | grep cat
bsachde+ 1435 1 0 18:34 ? 00:00:00 /usr/libexec/kde4/polkit-kde-authentication-agent-1
bsachde+ 3193 3165 0 19:54 pts/4 00:00:00 cat
bsachde+ 3195 1399 0 19:54 pts/1 00:00:00 grep --color=auto cat
[bsachdeva@localhost Tue Nov 25 18:34:06 IST 2014 ~]$
bsachdeva : cat - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Tue Nov 25 19:54:28 IST 2014 ~]$cat
-
bsachdeva : cat
bsachdeva : bash


Page not found - Mozilla Firefox
Konsole
libreoffice-writer
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Tue Nov 25 19:56:57 IST 2014 ~]$ps -ef | grep cat
bsachde+ 1435 1 0 18:34 ? 00:00:00 /usr/libexec/kde4/polkit-kde-authentication-agent-1
bsachde+ 3265 3237 0 19:57 pts/4 00:00:00 cat
bsachde+ 3267 3207 0 19:57 pts/1 00:00:00 grep --color=auto cat
[bsachdeva@localhost Tue Nov 25 19:56:57 IST 2014 ~]$kill -9 3265
[bsachdeva@localhost Tue Nov 25 19:56:57 IST 2014 ~]$
bsachdeva : bash - Konsole <2>
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Tue Nov 25 19:57:19 IST 2014 ~]$cat
Killed
[bsachdeva@localhost Tue Nov 25 19:57:19 IST 2014 ~]$
bsachdeva : bash
bsachdeva : bash


Page not found - Mozilla Firefox
Konsole
libreoffice-writer
08:00 PM
Local
bsachdeva : bash - Konsole
File Edit View Bookmarks Settings Help
[bsachdeva@localhost Tue Nov 25 19:56:57 IST 2014 ~]$cat
^Z
[1]+ Stopped cat
[bsachdeva@localhost Tue Nov 25 19:56:57 IST 2014 ~]$jobs
[1]+ Stopped cat
[bsachdeva@localhost Tue Nov 25 19:56:57 IST 2014 ~]$kill -9 %1
[1]+ Stopped cat
[bsachdeva@localhost Tue Nov 25 19:56:57 IST 2014 ~]$jobs
[1]+ Killed cat
[bsachdeva@localhost Tue Nov 25 19:56:57 IST 2014 ~]$kill -l
1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL 5) SIGTRAP
6) SIGABRT 7) SIGBUS 8) SIGFPE 9) SIGKILL 10) SIGUSR1
11) SIGSEGV 12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP
21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU 25) SIGXFSZ
26) SIGVTALRM 27) SIGPROF 28) SIGWINCH 29) SIGIO 30) SIGPWR
31) SIGSYS 34) SIGRTMIN 35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
[bsachdeva@localhost Tue Nov 25 19:56:57 IST 2014 ~]$
bsachdeva : bash

vi Introduction

"vi" (pronounced "vee eye") is a text editor with a deceptively simple appearance that belies its great power and efficiency. New users soon realize that there is far more to this little program than meets the eye.

vi, or one of its clones, is found in almost every version of Linux and Unix, and, in fact, it is the *only* editor that is available in virtually every Unix installation. Moreover, clones of vi are available for nearly every other operating system in use today, including Microsoft Windows. This makes it one of the most widely available programs in the world!

This near-ubiquity plus its power and efficiency (at least for experienced users) makes vi one of the most important programs for students of Linux to learn. And a command of vi helps differentiate a beginner from a true power user of Linux or Unix.

vi History

The vi editor was developed starting around 1976 by Bill Joy, who was then a graduate student at the University of California at Berkeley. Joy later went on to help found Sun Microsystems and became its Chief Scientist.

"ed" was the original Unix text editor. Like other early text editors, it was line oriented and used from dumb printing terminals. Joy first developed "ex" as an improved line editor that supported a superset of ed commands. He then developed vi as a "visual interface" to ex. That is, it allows text to be viewed on a full screen rather than only one line at a time. vi takes its name from this fact.

vi remains very popular today in spite of the development and widespread availability of GUI (graphical user interface) mode text editors which are far more intuitive and much easier for beginners to use than text-mode text editors such as vi. GUI-mode text editors include gedit and Emacs, both of which have become very common on Linux and other Unixes today.

vi Features

Many Linux users avoid using vi because of their bewildering and frustrating initial experience with it. vi is definitely not intuitive -- at least for neophytes. Particularly confusing to new users is their inability to simply begin typing away when vi starts, in contrast to conventional text editors and word processors.

However, vi gradually becomes easier to use after practice. In fact, its adherents claim that it is extremely fast and efficient once you become accustomed to it, and many people come to miss its power in situations where they have to use other text editors.

Although vi's role as the standard Linux and Unix editor may be to some extent an accident of history, this editor also has features which definitely merit its continued popularity and widespread use:

- It is present in almost every Linux Unix system, even the most minimal.

- It is very small. In fact, some versions have a total code size of less than 100KB. This makes it easy to include vi on even the tiniest versions of Linux, such as those in embedded systems and those that run from a single floppy disk.

- It is typist-friendly, at least once you get used to it. For example, the commands are very short, usually just a few keystrokes. And because vi does not use the mouse, there is never any need to remove one's hands from the keyboard. This can speed up editing substantially.

- It is very powerful, as just a few very short commands can make sweeping changes to large documents. In fact, vi is more powerful than most of its users realize, and few of them know more than just fraction of all the commands.

Even if you are not going to use it as your main editor, becoming familiar with vi can still be a good investment. For example, many Unix tools, applications and games employ a subset of vi's commands. This includes some of the best diagnostic and forensic tools, even those for use with other operating systems such as Microsoft Windows. Moreover, vi is often the only editor still available for use while trying to repair a damaged Linux or Unix system that retains only minimal functionality.

vi has a total of approximately 150 basic commands. However, it is only necessary to know about a dozen of them in order to be able to use it effectively. And you can become really fast after you have learned about two dozen. This tutorial covers enough commands to make you a fairly accomplished vi user and set you on your way to becoming a true Linux/Unix geek.

Opening and Closing Files

vi can be used both when your system is in text mode (the entire screen is devoted to text and there are no images) and when your system is in GUI mode (the screen contains windows, images and menus). When it is in GUI mode (usually KDE or Gnome), vi runs in a terminal window. A terminal window is a text-only window, and it can usually be opened by clicking on an icon (small image) of a computer screen.

(In the case of Red Hat Linux, the terminal window can be opened by clicking on the icon of a red hat in the lower left hand corner of the screen, opening the System Tools menu and then selecting Terminal from that menu. It can be convenient to add the icon for the terminal window to the launcher panel along the bottom of the screen, if it is not already there.)

There are at least two ways to use vi to simultaneously create and open a new file. One is by just typing vi at the command line, like this:

```
vi
```

This creates an empty file that will not have a name until you save its contents to disk (i.e., transfer the text you typed into it to your hard disk, floppy disk, etc. for long term storage).

You will notice that a (mostly) blank window opens after you type this command. Do not be concerned

about trying to type any text into the screen yet (although you will probably want to). This topic will be covered in the next chapter.

(You might initially see something different than the nearly empty screen described above. The reason is most likely that your computer is using a clone of vi instead of the real thing. This is no problem, as the core commands of vi are also used in the clones. If Vim, the most popular clone, is installed in your system, the initial screen will have some text in the center that begins with something like "VIM - Vi IMproved." vi clones will be discussed in more detail in a later section.)

A second way to open a new file is by typing vi followed by the name of the file to be created, for example:

```
vi apple
```

This creates a new file named "apple" in the current directory (the directory or folder which is currently open on your all-text screen or your terminal window).

If you want, you could create the same file with an extension such as ".txt" added to the end of the file name. In Linux this is merely a matter of convenience (or habit), and it generally makes no real difference for the file because it remains a plain text file in either case. For example:

```
vi apple.txt
```

New files can also be created in directories other than the current directory. For example, to create and open a file named "apple" in the directory /home/jane/, type:

```
vi /home/jane/apple
```

(As you probably already know from your study of Linux or Unix, the above example can be performed regardless of where your current directory is on your system because the path for the command begins with a root directory, i.e. /home. This example assumes that you have the proper permissions to create files in the /jane directory.)

The technique for opening existing files is identical. Just type vi followed by the name of the file. For example, to open an existing file named "pear" in the directory /usr/local/, type:

```
vi /usr/local/pear
```

When you open an existing or new file, you will notice that nearly the entire screen is either devoted to the text of the file or available for entering text. The only exception is the very bottom line, which provides information about the file. This information initially includes the number of lines and the number of characters in the file. When commands are entered, the bottom line is used instead to show the entry of such commands.

You will also notice a blinking black rectangle in the upper left hand corner of the screen. That is the cursor, and its role is to tell you what character or position in the file is currently ready to be acted upon.

Stretching down the left hand edge of the screen is a column of tildes (~). These are merely row markers and are not part of the text. They will disappear as your typing extends down the screen.

To close a file to which no changes have been made, hit ESC (the Esc key, which is located in the upper left hand corner of the keyboard), then type :q (a colon followed by a lower case "q") and finally press ENTER. (The term "hit" is used here instead of "press" to emphasize that it is not necessary to keep the ESC key held down but just to press it momentarily.)

To close a file to which changes have been made (such as text having been added or removed) without saving the changes, hit ESC, type :q! and then press ENTER. This is sometimes referred to as a "forced quit."

vi works with a buffer (a block of memory in the RAM chips). When you open an existing file, vi copies that file from the hard disk (or floppy, CDROM, etc.) to a buffer. All changes that you make to a file are initially made only to the copy in the buffer, and they are only made to the file itself when you "save" your changes. "Saving" a file means writing (i.e., transferring) the contents of the buffer to the hard disk (or floppy disk).

Likewise when you open a new file. All text you enter (and subsequent edits you make to it) exists only in the buffer until you save the file to disk.

To save the changes that have been made to a file, hit ESC, type :qw and then press ENTER. The "w" stands for "write." An alternative, and perhaps easier, way to save a file and quit at the same time is to hit ESC and then type ZZ (two capital Z's in succession).

After you have created a new text file and closed it, you might want to confirm that nothing went wrong and that the file actually exists. Probably the simplest way to do this is to use the standard Unix ls command, which displays a list of all of the files in the current directory.

Entering Text

vi has two basic modes of operation: command mode and text insert mode. How to switch back and forth between them is probably the most confusing thing about vi for beginners. But it is actually very simple, and once you get used to it you might also find it quite efficient.

Command mode is the default mode when a file (existing or new) is opened. (This is the opposite of most text and word processors and therefore may seem counter-intuitive.) Because every file opens initially in command mode, you can not immediately begin typing text. That is, everything that is typed on the keyboard is interpreted by vi to be a command.

Examples of the many types of commands that you can perform on a file while in command mode are (1) switching to text insert mode, (2) moving the cursor around the file, (3) deleting characters or lines, (4) transposing characters, (6) changing case, (6) appending the contents of the file to another (closed) file, (7) setting vi options, (8) saving the file to disk and (9) closing the file and quitting vi.

The other mode, text insert mode, is also referred to as simply "insert mode" or "input mode." It is used for entering text into the buffer memory (and simultaneously onto the screen). In this mode everything

that is typed on the keyboard is added to the text and does not become a command (although you can perform some command operations in text mode with vi clones).

The most common way to switch from command mode to the input mode is to use the i (which stands for "insert" or "input") command. This is accomplished by simply typing the letter i while in command mode. Now you are ready to start typing text.

Unlike word processors and even most word editors, there is no automatic word wrap in the traditional version of vi (although you will notice it in some clones). New lines are started by pressing ENTER.

When you are finished typing text or you want to perform some other operation such as moving to a different position in the text or deleting some of it, hit ESC in order to return to the command mode.

It is not easy to determine vi's current mode. But this is not really a problem because, if in doubt, just press ESC and you will know that you are in the command mode (this will put you in command mode if you were not already in it).

Once you have typed some text, you can use the four basic commands for moving the cursor around the text. These commands enable you to go to any desired location in order to modify the text, including making insertions and deletions. The four basic cursor positioning commands are:

h move cursor one character to left

j move cursor one line down

k move cursor one line up

l move cursor one character to right

Each of these commands can be either used by itself or modified by typing an integer in front of it to indicate the number of characters or lines to move. For example, typing (in command mode, of course)

3j

will move the cursor down three lines. Or typing 2h will move it two characters to the left.

These commands can be repeated by holding the key down. If you attempt an impossible movement, such as pressing k when the cursor is on the top line, the screen might flash or a beeping sound might be made (depending on how your computer is set up).

The cursor can be moved directly to any desired line by using the G command preceded by the line number. For example, typing

5G

moves the cursor to the fifth line from the top of the text. Just typing G without any number moves the cursor to the final line of text.

When you switch from command mode to input mode with the i command and then start typing text, each character you type is placed to the left of the character covered by the cursor. This causes the character covered by the cursor as well as everything to its right to be shifted to the right.

There will be times when you will want to place a character to the right of the character under the cursor. This is particularly useful when the cursor is over the last character in a line and you want to append the line. To do this, simply use the a (lower case "a," which stands for "append") command instead of the i command to switch from command mode into insert mode.

After you have saved a file that you have created or modified using vi, you might want to verify that its contents are really what you had intended. One way to do this is to use cat, the Unix concatenation utility. (No, this has no relationship to the popular domesticated animal whose name has the same spelling). For example, type:

```
cat /home/john/fruit/lemon
```

This causes the contents of the file "lemon" in the /home/john/fruit/ directory to be displayed on the screen in a read-only form.

Of course, if your current directory is already /home/john/fruit/, then you only need to type:

```
cat lemon
```

Editing Text

vi offers a rich assortment of commands for editing text. Among the most basic are those used for deleting or erasing.

The x (lower case "x") command deletes the character immediately under (i,e., covered by) the cursor. To delete any desired character, just switch to the command mode (if you are not already there) and then use an appropriate combination of the h, j, k and l commands (of course, one at a time) to move the cursor to that character. Then type x and the character is deleted.

By pressing x continuously instead of just hitting it once, the cursor continuously moves to the right and each character under it is successively deleted.

The X (upper case "X") command is similar except that it deletes the character to the left of the cursor rather than the character under it.

There are several additional commands for deleting text. The D (upper case "D") command removes the text on the current line from the character under the cursor to the end of the line.

The d (lower case "d") command is very flexible because it can be modified to delete any number of characters, words or lines. Typing d by itself will not do anything, but typing dw causes the character the cursor is resting on and the remaining characters to the right of it in the same word to be deleted. (The "w" stands for "word.")

Typing 2dw causes the character under the cursor, the remaining characters to the right of it in the same word and all of the characters in the next word to be deleted. For example, typing 2dw with the cursor on the "a" of the string "pineapple plantation" causes the string "apple plantation" to be deleted.

As another example, typing 3dw with the cursor on the "j" of the string "the bluejay flew south" causes the string "jay flew south" to be deleted. That is, "jay" and two words to the right of it are deleted.

Deleting an entire line can be accomplished with the dd command. This command can also be used to delete multiple lines by preceding it with an integer representing the number of lines to be removed. For example, typing 2dd will delete two consecutive lines beginning with the current line.

With some terminals, deletion of a line causes it to be replaced on the screen with an "@" character. This character merely represents an empty line and is not inserted into the text. Its purpose is to relieve the processor from having to redraw the screen (i.e., change the whole screen). This character can be removed if desired by typing r (or l on some terminals) while holding down the CTRL key.

The change command c (lower case "c") differs from the delete command in that it not only deletes a section of text but also activates insert mode to allow you to type in replacement text. After you have completed typing in the replacement text, be sure to press ESC to return to the command mode.

As is the case with d, the c command is not used by itself but is only used in combination with another letter after it and an optional integer before it.

For example, the command cw (which stands for "change word") deletes the characters in the current word under and to the right of the cursor and then switches vi to the insert mode so that you can enter text to replace the deleted characters. The number of new characters typed in can be the same as, fewer or more than the number deleted.

The amount of text to be changed can be increased by preceding the command with a number. For instance, typing 2cw will additionally remove the next word for replacement with whatever is typed in. The space between the words is not preserved.

The d and c commands can also be modified by other characters in addition to "w." For example they can be used with "b," which stands for "back." Thus, typing 3bd will delete the characters to the left of the cursor in the current word together with the two words to the left of the current word.

The cc command erases the current line, leaving it blank and awaiting replacement text. Preceding this command with an integer will delete that number of lines, beginning with the current line. For example, typing 5cc will allow you to change five consecutive lines starting with the current line.

Another change command, R, differs from the c commands in that it does not initially delete anything. Rather, it activates insert mode and lets you replace the characters under the cursor one at a time with characters that you type in.

vi supports several types of transposition. Transposing the order of two adjacent characters is easy with the xp command. Just place the cursor on the left-most of the two characters, type x to erase the left character and then type p for the deleted character to be put to the right of the cursor.

Two adjacent words can be transposed with the deep command. To use it, position the cursor in the space just to the left of the word on the left and type deep. Two adjacent lines can be transposed with the ddp command by placing the cursor on the upper line and typing ddp.

It is also a simple matter to change the case of a letter. When the cursor is over the desired letter, hit the "~" (tilde) key. This will change a capital letter to a small letter and visa versa.

The J (upper case "J") command is used to join the next line to the current line. The opposite operation, splitting a line, is accomplished in insert mode by merely positioning the cursor over what will be the first character of the new line and then hitting ENTER.

vi also has an undo capability. The u (lower case "u") command is used to reverse the effects of an already issued command that has changed the buffer, but which is not yet written to disk. U (upper case "U") undoes all of the changes that have been made to the current line during your current visit to it.

Searching and Replacing

vi also has powerful search and replace capabilities. To search the text of an open file for a specific string (combination of characters or words), in the command mode type a colon (:), "s," forward slash (/) and the search string itself. What you type will appear on the bottom line of the display screen. Finally, press ENTER, and the matching area of the text will be highlighted, if it exists. If the matching string is on an area of text that is not currently displayed on the screen, the text will scroll to show that area.

The formal syntax for searching is:

```
:s/string
```

For example, suppose you want to search some text for the string "cherry." Type the following and press ENTER:

```
:s/cherry
```

The first match for "cherry" in your text will then be highlighted. To see if there are additional occurrences of the same string in the text, type n, and the highlight will switch to the next match, if one exists.

The syntax for replacing one string with another string in the current line is

```
:s/pattern/replace/
```

Here "pattern" represents the old string and "replace" represents the new string. For example, to replace each occurrence of the word "lemon" in a line with "orange," type:

```
:s/lemon/orange/
```

The syntax for replacing every occurrence of a string in the entire text is similar. The only difference is

the addition of a "%" in front of the "s":

:%s/pattern/replace/

Thus repeating the previous example for the entire text instead of just for a single line would be:\

:%s/lemon/orange/

Working With Multiple Files

It is easy to insert text into an open file from another file. All that is necessary is to move the cursor to the location where you want the text inserted, then type

:r filename

where "filename" is the name of the file to insert.

For example, if you want to copy the contents of the file "peach" into the file "fruit," you would first position the cursor to the desired line in "fruit" and then type

:r peach

Notice that this operation causes no change to the file "peach."

You can also append text from the currently open file to any other file. This is accomplished using the :w (colon + "w") command followed without a space by >>. For example, to append the contents of a currently open file named "pear" to the file named "apple," type

:w>> apple

At times it can be convenient to open multiple files simultaneously. This is efficiently accomplished by just listing all of the files to be opened after the vi command. For example, to simultaneously open files about three kinds of fruit, type:

vi apple pear orange

This allows you to edit "apple" first. After saving "apple," typing :n calls up "pear" for editing.

If you want to simultaneously open all files in the current directory, just type vi * (vi + space + asterisk).

Additional Operations

A careful study of the preceding material is sufficient for you to be well on your way to becoming proficient at vi. All you really need to do now is continue to practice what you have learned. The following material is presented for those of you who want to learn just a few more useful commands.

As you have learned, creating and opening files in vi can be a very simple matter. However, many combinations of options are available that can add much power and flexibility for these tasks, as can be seen by looking at the full syntax for opening files:

```
vi [flags] [cmd] [filename]
```

The square brackets ([ ]) around each section of arguments (modifiers) of the command indicates that they are optional. (That is, a file can be opened by just typing vi alone or by typing it with any combination of the three arguments. For instance, the example of vi dog contains only the mandatory vi and the optional third argument, which is the name of the file to open.)

As only one of many possible examples of adding options for opening files, an existing file can be opened with the cursor appearing on any desired line instead of just on the first line. (One situation in which this can be particularly useful is if your file is part of a program which you are writing and the compiler reports an error on a specific line in that file.) This is accomplished by adding the + (plus sign) command followed the desired line number. For example, to open the file "apple" with the cursor located on the third line, type:

```
vi +3 apple
```

Use of the + command without any modifying number opens a file with the cursor positioned on the last line of text. This can save some keystrokes when you want to open a file just to append data to the end of it. For example:

```
vi + apple
```

You have already learned several commands for switching from command mode to insert mode, including i for inserting to the left of the cursor position, a for inserting to the right of the cursor position and the c commands for changing text. A more complete list is as follows:

a	appends after current cursor position
A	appends at end of current line
c	starts a change option
C	starts a change option from current position to end of current line
i	inserts to the left of the cursor position
I	inserts at start of line
o	cursor moves to new, blank line below its current position
O	cursor moves to new, blank line above its current position
R	replaces characters one at a time

A simple way to obtain basic information about any file that is currently open, including name, size and the current line number, is to hold down CTRL and type g. This data appears on the bottom line of the display.

Summary of Commands

The following list contains the basic commands presented in the first eight pages of this tutorial along with occasional examples of usage (shown in parenthesis). They are presented in roughly the same order in which they appear in the tutorial. (All commands that begin with a colon are followed by ENTER.)

vi typed at the command line to open one or more files in the same directory
(`vi tomato.txt` opens a file named "tomato.txt" in the current directory)
(`vi parsley sage rosemary` opens the three files "parsley," "sage" and "rosemary" in the current directory)

vi * typed at the command line to open every file in the current directory

:q closes (quits) a file to which no changes have been made

:q! quits without saving any changes

:w writes (i.e., saves) the current file to disk

:wq writes the buffer contents to disk (i.e., saves changes) and quits

ZZ same as **:wq**

i activates text insert mode, inserting text immediately under the current position of the cursor

h moves the cursor one character to the left
(`2h` moves the cursor two characters to the left)

j moves the cursor one line down
(`3j` moves the cursor three lines down)

k moves the cursor one line up

l moves the cursor one character to the right

G moves the cursor to the desired line; moves the cursor to the last line of text if not preceded by a modifying integer
(`5G` moves the cursor to the fifth line)

a switches to insert mode and allows insertion of text immediately to the right of the cursor

x deletes the character immediately under the cursor
(`xxx` deletes the character immediately under cursor and then deletes the two characters to its right)

X deletes a single character to the left of cursor

D removes the text on the current line from the character under the cursor to the end of the line

dw deletes the character immediately under the cursor and the remaining characters to the right of it in the same word

(`2dw` deletes the character immediately under the cursor, the remaining characters to the right of it in same word and all of the next word)

dd deletes the entire line containing the cursor, and the cursor then moves to the next line

(`2dd` deletes two consecutive lines beginning with the current line)

cw deletes the character under the cursor and to its right in the same word and allows new characters to be typed in to replace them

(`2cw` deletes the character under the cursor and to its right in the same word and in the next word, and then allows replacement characters to be typed in)

cc erases the current line and allows replacement text to be typed in

(`2cc` erases the current line and the next line and allows replacement text to be typed in for both lines)

cb deletes the characters to the left of the cursor in the current word and allows replacement characters to be typed in

(`3cb` deletes the characters to the left of the cursor in the current word together with the two words to its left and then allows replacement text to be typed in)

R activates text input mode allowing text under and to the right of the cursor to be overwritten one character at a time

xp transposes two adjacent characters

deep transposes two adjacent words

ddp transposes two adjacent lines

~ changes case of the character under the cursor

J joins the current line with the next line

u reverses the effects of the most recent command that has changed the buffer

U undoes all changes made to the current line during the current visit to it

:s/ searches the text for the first instance of a designated string

(`:s/cucumber` searches the text for the first instance of the string "cucumber")

n searches the text for the next instance of a designated string

:s/ / / replaces the first instance of a designated string

(`:s/cucumber/radish/` replaces the first instance of the string "cucumber" with the string "radish")

:%s/ / / replaces every instance of a designated string

(`:%s/cucumber/radish/` replaces every instance of the string "cucumber" with the string "radish")

:r inserts text into the currently open file from another file

(`:r lettuce.txt` inserts text into the currently open file from the file named "lettuce.txt")

:w>> appends the text from the currently open file into another file

(`:w>> cabbage` appends the text from the currently open file into the file named "cabbage")

Additional vi Resources

There may be several good sources of information about vi or its clones already residing in your computer. One of them is the vi manual pages, which are accessed by typing man vi at the command line.

Another is the help files, which can be accessed while vi is in use by typing :help and pressing ENTER. To exit the help screen type :q and press ENTER.

Vim comes with a tutorial which is stored in text form in a directory such as /usr/share/vim/vim74/tutor/. After confirming the exact location on your system, you can navigate to that directory and open the file with a command such as

```
cat tutor | more
```

This assumes that the tutorial is named "tutor" in your system. The "cat" command is used to open the tutorial in read-only mode. The "|" and the "more" are standard Unix commands that work together to display a long file one page at a time for easier reading.

nvi comes with a tutorial which is stored as a compressed file in /usr/doc/nvi/. This file can be accessed by copying it to your home directory, uncompressing it and then opening it with vi. For example, from your home directory type:

```
cp /usr/doc/nvi/vi.beginner.gz

gunzip vi.beginner.gz

vi vi.beginner
```

An advanced tutorial is also available in the same directory.

Introduction to command line arguments and logical statements in vi-editor

Arithmetic operations.

Syntax:
expr op1 math-operator op2

Examples:
$ expr 1 + 3
$ expr 2 - 1
$ expr 10 / 2
$ expr 20 % 3
$ expr 10 * 3
$ echo `expr 6 + 3`
Note:
expr 20 %3 - Remainder read as 20 mod 3 and remainder is 2.
expr 10 * 3 - Multiplication use * and not * since its wild card.
For the last statement not the following points
(1) First, before expr keyword we used ` (back quote) sign not the (single quote i.e. ') sign. Back quote is generally found on the key under tilde (~) on PC keyboard OR to the above of TAB key.
(2) Second, expr is also end with ` i.e. back quote.
(3) Here expr 6 + 3 is evaluated to 9, then echo command prints 9 as sum
(4) Here if you use double quote or single quote, it will NOT work
For e.g.
$ echo "expr 6 + 3" # It will print expr 6 + 3
$ echo 'expr 6 + 3' # It will print expr 6 + 3

- *The read Statement*
  Use to get input (data from user) from keyboard and store (data) to variable.
  Syntax:
  read variable1, variable2,...variableN
  Following script first ask user, name and then waits to enter name from the user via keyboard. Then user enters name from keyboard (after giving name you have to press ENTER key) and entered name through keyboard is stored (assigned) to variable fname.
  $ vi sayH
  #
  #Script to read your name from key-board
  #
  echo "Your first name please:"
  read fname
  echo "Hello $fname, Lets be friend!"
- Run it as follows:
  $ chmod 755 sayH
  $ ./sayH
  Your first name please: ***world***

Hello world, Lets be friend!

- *if and test command or [expr]*
  test command or [expr] is used to see if an expression is true, and if it is true it return zero(0), otherwise returns nonzero for false.
  Syntax:
  test expression OR [expression]

  Example:
  Following script determine whether given argument number is positive.
  $ cat > ispostive
  #!/bin/sh
  #
  # Script to see whether argument is positive
  #
  if test $1 -gt 0
  then
  echo "$1 number is positive"
  fi
- Run it as follows
  $ chmod 755 ispostive
  $ ispostive 5
  5 number is positive
  $ispostive -45
  Nothing is printed
  $ispostive
  ./ispostive: test: -gt: unary operator expected
  Detailed explanation
  The line, if test $1 -gt 0 , test to see if first command line argument($1) is greater than 0. If it is true(0) then test will return 0 and output will printed as 5 number is positive but for -45 argument there is no output because our condition is not true(0) (no -45 is not greater than 0) hence echo statement is skipped. And for last statement we have not supplied any argument hence error ./ispostive: test: -gt: unary operator expected, is generated by shell , to avoid such error we can test whether command line argument is supplied or not.
  test or [expr] works with
  1.Integer (Number without decimal point)
  2.File types
  3.Character strings
  For Mathematics, use following operator in Shell Script

Mathematical Operator in Shell Script	**Meaning**	**Normal Arithmetical/ Mathematical Statements**	**But in Shell**	
			For test statement with if command	**For [expr] statement with if command**
-eq	is equal to	5 == 6	if test 5 -eq 6	if [5 -eq 6]

-ne	is not equal to	5 != 6	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if [5 -ge 6]

NOTE: == is equal, != is not equal.

For string Comparisons use

Operator	Meaning
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
string1	string1 is NOT NULL or not defined
-n string1	string1 is NOT NULL and does exist
-z string1	string1 is NULL and does exist

Shell also test for file and directory types

Test	Meaning
-s file	Non empty file
-f file	Is File exist or normal file and not a directory
-d dir	Is Directory exist and not a file
-w file	Is writeable file
-r file	Is read-only file
-x file	Is file is executable

Logical Operators

Logical operators are used to combine two or more condition at a time

Operator	Meaning
! expression	Logical NOT
expression1 -a expression2	Logical AND
expression1 -o expression2	Logical OR

- *if...else...fi*

  If given condition is true then command1 is executed otherwise command2 is executed.

  Syntax:

  if condition
  then
  condition is zero (true - 0)
  execute all commands up to else statement

```
else
        if condition is not true then
        execute all commands up to fi
fi
```

For e.g. Write Script as follows:

- $ vi isnump_n

```
#!/bin/sh
#
# Script to see whether argument is positive or negative
#
if [ $# -eq 0 ]
then
echo "$0 : You must give/supply one integers"
exit 1
fi

if test $1 -gt 0
then
echo "$1 number is positive"
else
echo "$1 number is negative"
fi
```

- Try it as follows:

$ chmod 755 isnump_n

$ isnump_n 5
5 number is positive

$ isnump_n -45
-45 number is negative

$ isnump_n
./ispos_n : You must give/supply one integers

$ isnump_n 0
0 number is negative

Detailed explanation

First script checks whether command line argument is given or not, if not given then it print error message as "*./ispos_n : You must give/supply one integers*". if statement checks whether number of argument ($#) passed to script is not equal (-eq) to 0, if we passed any argument to script then this if statement is false and if no command line argument is given then this if statement is true.

The last sample run **$ isnump_n 0** , gives output as "*0 number is negative*", because given argument is not > 0, hence condition is false and it's taken as negative number. To avoid this replace second if statement with **if test $1 -ge 0**.

- *Multilevel if-then-else*

Syntax:

```
if condition
```

```
then
        condition is zero (true - 0)
        execute all commands up to elif statement
elif condition1
then
        condition1 is zero (true - 0)
       execute all commands up to elif statement
elif condition2
then
        condition2 is zero (true - 0)
        execute all commands up to elif statement
else
      None of the above condtion,condtion1,condtion2 are true (i.e.
        all of the above nonzero or false)
        execute all commands up to fi
fi
```

- *for Loop*

  Syntax:

  for { variable name } in { list }

  do

  execute one for each item in the list until the list is

  not finished (And repeat all statement between do and done)

  done

  Even you can use following syntax:

  Syntax:

```
for (( expr1; expr2; expr3 ))
      do
          .....
                ...
          repeat all statements between do and
          done until expr2 is TRUE
      Done
```

  In above syntax BEFORE the first iteration, ***expr1*** is evaluated. This is usually used to initialize variables for the loop.

  All the statements between do and done is executed repeatedly UNTIL the value of ***expr2*** is TRUE.

  AFTER each iteration of the loop, ***expr3*** is evaluated. This is usually use to increment a loop counter.

```
$ cat > testfor
for i in 1 2 3 4 5
do
echo "Welcome $i times"
done
```

- Run it above script as follows:

  $ chmod +x testfor

  $ ./testfor

```
$ cat > for2
for (( i = 0 ; i <= 5;i++ ))
```

```
do
 echo "Welcome $i times"
done
```

- Run the above script as follows:

$ chmod +x for2
$ ./for2
Welcome 0 times
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times

In above example, first expression (i = 0), is used to set the value variable **i** to zero.
Second expression is condition i.e. all statements between do and done executed as long as expression 2 (i.e continue as long as the value of variable **i** is less than or equel to 5) is TRUE.
Last expression **i++** increments the value of **i** by 1 i.e. it's equivalent to i = i + 1 statement.

- *while loop*

Syntax:
while [condition]
do
command1
command2
command3
..
....
done

Loop is executed as long as given condition is true. For e.g.:

```
$cat > nt1
#!/bin/sh
#
#Script to test while statement
#
#
if [ $# -eq 0 ]
then
   echo "Error - Number missing form command line argument"
   echo "Syntax : $0 number"
   echo " Use to print multiplication table for given number"
exit 1
fi
n=$1
i=1
while [ $i -le 10 ]
do
  echo "$n * $i = `expr $i \* $n`"
  i=`expr $i + 1`
done
```

- Save it and try as

$ chmod 755 nt1
$./nt1 7

- *The case Statement*

  The case statement is good alternative to Multilevel if-then-else-fi statement. It enable you to match several values against one variable. Its easier to read and write.

  Syntax:

  case $variable-name in
  pattern1) command
  ...
  ..
  command;;
  pattern2) command
  ...
  ..
  command;;
  patternN) command
  ...
  ..
  command;;
  **) command*
  ...
  ..
  command;;
  esac

  The *$variable-name* is compared against the patterns until a match is found. The shell then executes all the statements up to the two semicolons that are next to each other. The default is *) and its executed if no match is found. For e.g. write script as follows:

- $ cat > car
  #
  # if no vehicle name is given
  # i.e. -z $1 is defined and it is NULL
  #
  # if no command line arg
- if [-z $1]
  then
  rental="*** Unknown vehicle ***"
  elif [-n $1]
  then
  # otherwise make first arg as rental
  rental=$1
  fi
  case $rental in
  "car") echo "For $rental Rs.20 per k/m";;
  "van") echo "For $rental Rs.10 per k/m";;
  "jeep") echo "For $rental Rs.5 per k/m";;
  "bicycle") echo "For $rental 20 paisa per k/m";;
  “*”) echo "Sorry, I can not gat a $rental for you";;
  esac

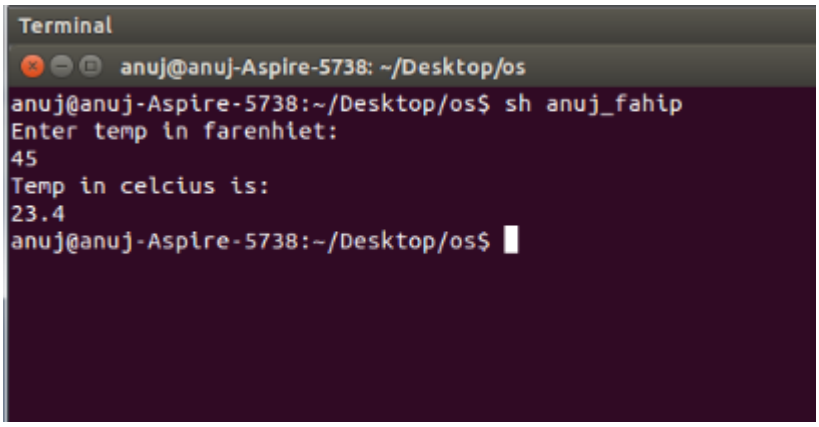
To implement following programs in vi-editor

1) To input length, breadth and radius and calculate area, perimeter and circumference.

```
echo "Enter the lenght: "
read len
echo "Enter the breadth: "
read br
echo "Enter radius of radius: "
read radius

echo "Area rectangle: " `expr $len*$br | bc`
echo "Perimeter rec: " `expr 2*\($len+$br\) | bc`
echo "Area circle: " `expr 3.14*$radius*$radius | bc`
```

2)To change temperature from Farenhiet to Celcius

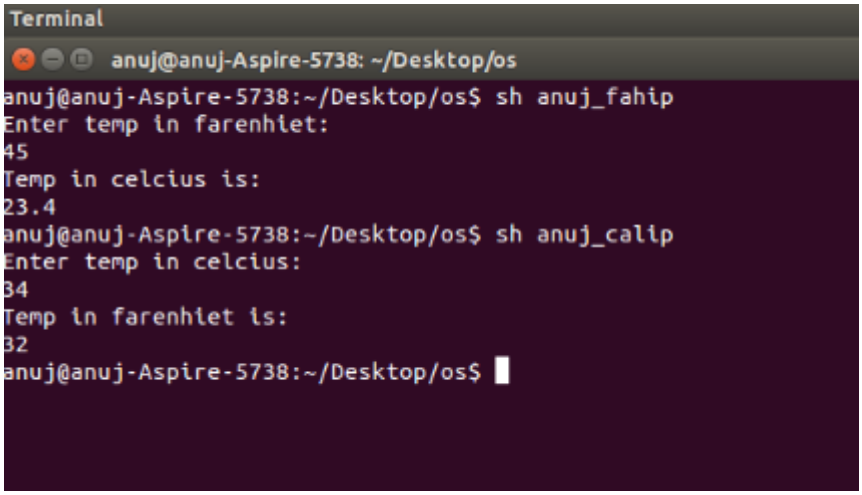
```
echo "Enter temp in farenhiet: "
read f
cel=`expr 1.8*\($f-32\)|bc`
echo "Temp in celcius is: "
echo $cel
```



3)To change temperature from Celcius to Farenhiet

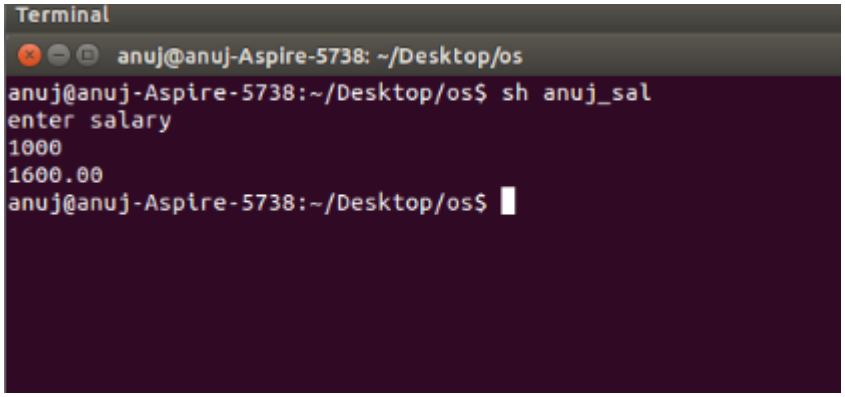
```
echo "Enter temp in celcius: "
read c
```

```
far=`expr \(5.0/9.0\)*$c+32|bc`
echo "Temp in farenhiet is: "
echo $far
```



4) To find total salary if DA is 40% of basic pay and Hr is 20% of basic pay

```
echo "enter salary"
read sal
total=`echo $sal+0.40*$sal+0.20*$sal|bc`
echo $total
```



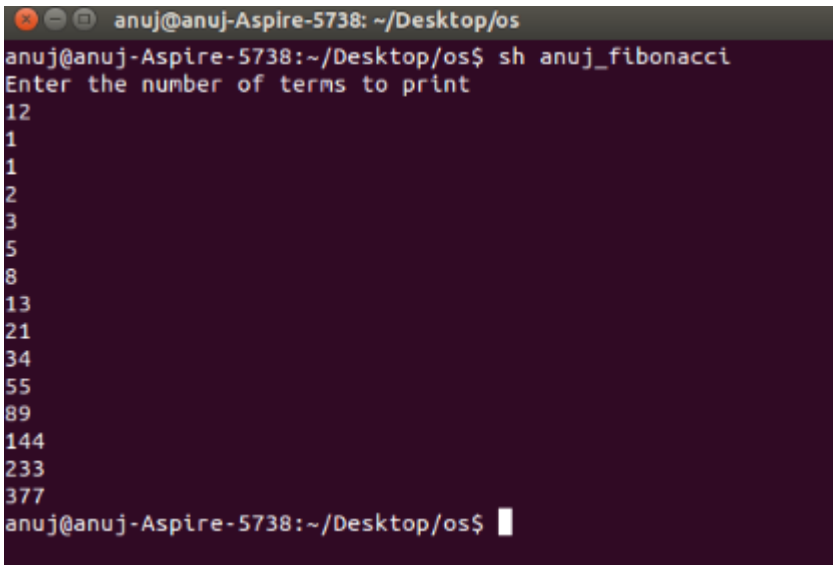
To implement following programs in vi-editor

1)To find factorial of a given number

```
echo "enter number"
read n
result=1
for i in $(seq 1 1 $n)
do
result=`echo $result*$i|bc`
done
echo "factorial of num  "$n" is "$result""
```

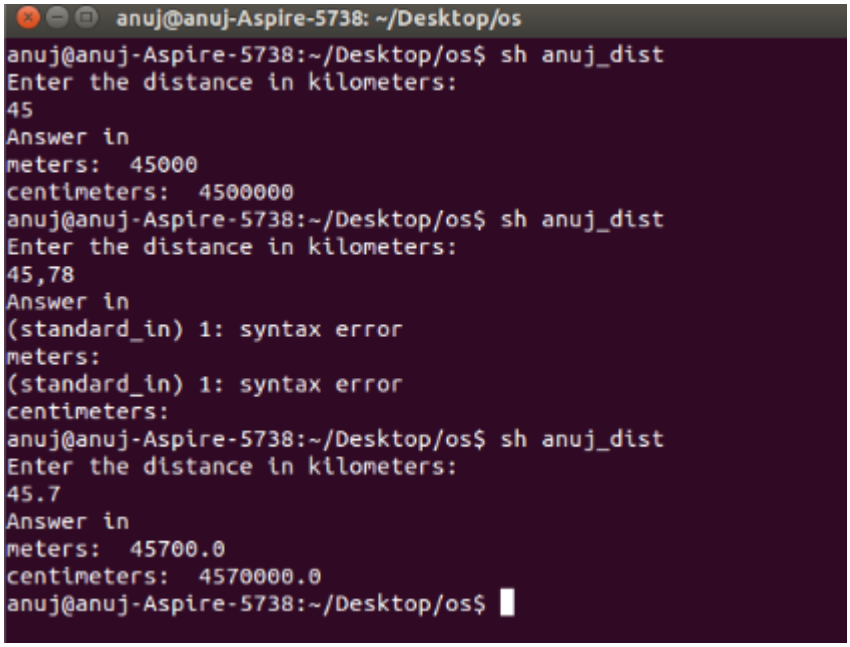
2) Generate Fibonacci series upto given number

```
echo "Enter the number of terms to print"
read terms
a=1
b=1
echo $a
echo $b
for i in $(seq 1 1 $terms)
do
   t=$a
   a=$b
   b=`expr $b+$t | bc`
   echo $b
done
```



3) To input distance in kilometers and convert it into meters and centimeters

```
echo "Enter the distance in kilometers: "
read kilom
echo "Answer in "
echo "meters: " `expr 1000*$kilom| bc`
echo "centimeters: " `expr $kilom*100000 | bc`
```



4) To generate prime numbers in the range 1-300

```
# Find prime nos from 1 to 300

for i in $(seq 2 1 300)
do
   isprime=1
   till=`expr $i - 1 | bc`
   for j in $(seq 2 1 $till)
   do
      mod=`expr $i % $j | bc`
      if [ $mod -eq 0 ]
      then
         isprime=0
         break
      fi
   done
   if [ $isprime -eq 1 -o $i -eq 2 ]
```

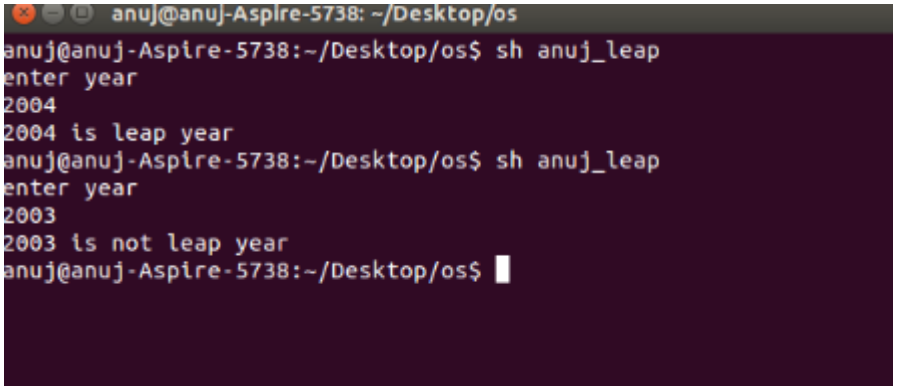
```
    then
        echo $i "number is prime"
    fi
done
```

```
anuj@anuj-Aspire-5738:~/Desktop/os$ sh anuj_prime
2 number is prime
3 number is prime
5 number is prime
7 number is prime
11 number is prime
13 number is prime
17 number is prime
19 number is prime
23 number is prime
29 number is prime
31 number is prime
37 number is prime
41 number is prime
43 number is prime
47 number is prime
53 number is prime
59 number is prime
61 number is prime
67 number is prime
71 number is prime
73 number is prime
79 number is prime
```

To implement following programs in vi-editor

1)To input an year and tell whether it is leap year or not

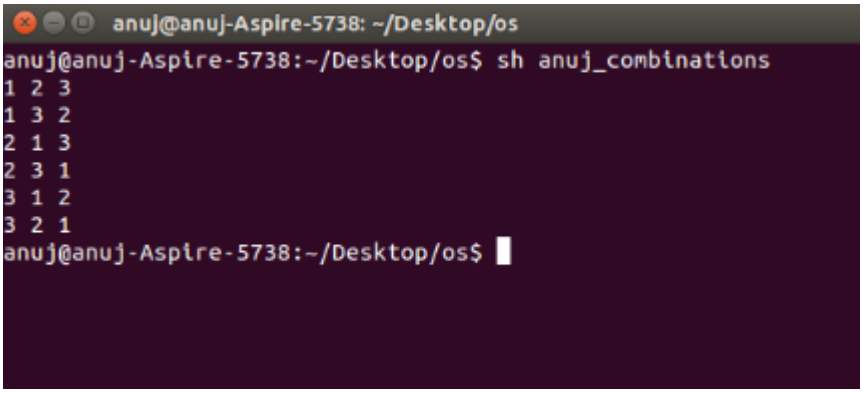
```
echo "enter year "
read year
year4=`echo $year%4|bc`
year400=`echo $year%400|bc`
if [ $year4 eq 0 a $year400 ne 0 ]
then
echo "$year" "is leap year""
else
echo "$year" "is not leap year""
fi
```



2) To generate all combinations of 1, 2 and 3

```
for i in $(seq 1 1 3)
do
for j in $(seq 1 1 3)
do
if [ $i eq $j ]
then
continue
else
for k in $(seq 1 1 3)
do
if [ $i ne $k a $j ne $k ]
then
echo $i $j $k
fi
```

```
done
fi
done
done
```

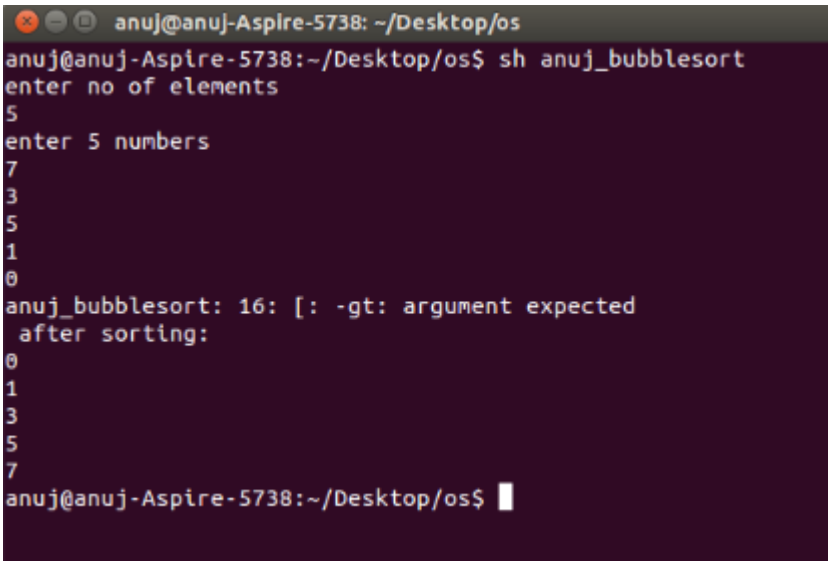


3) To find maximum and minimum in a given set of numbers

```
min=$1
max=$1
for i in $*
do
if test $min gt $i
then
min=$i
fi
if test $max lt $i
then
max=$i
fi
done
echo "minimum:"$min
echo "maximum:"$max
```

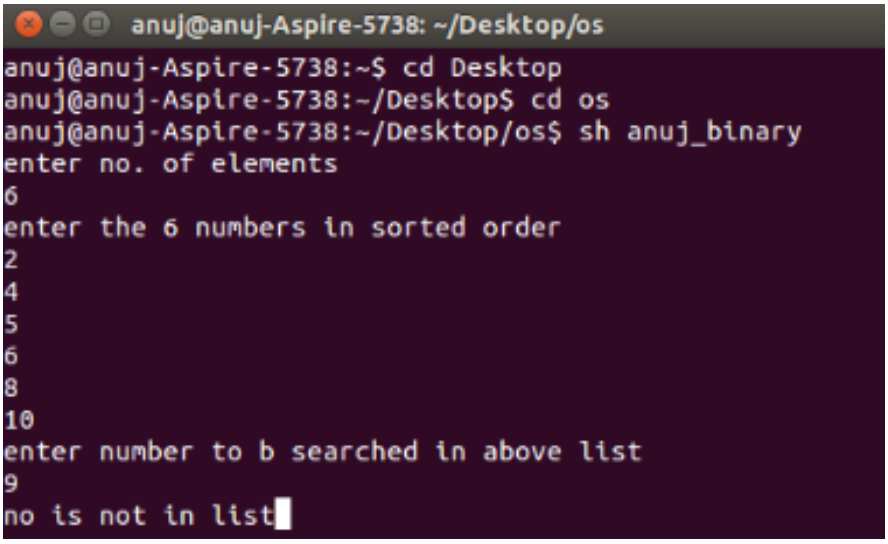
4)To implement Bubble sort

```
echo "enter no of elements"
read n
echo "enter $n numbers"
for i in $(seq 1 1 $n)
do
read x$i
done
for i in $(seq $n 1 1)
do
for j in $(seq 1 1 $i)
do
j1=`echo $j+1|bc`
eval temp1=\$x$j1
eval temp2=\$x$j
if [ $temp2 gt $temp1 ]
then
eval x$j1=$temp2
eval x$j=$temp1
fi
done
done
echo " after sorting: "
for k in $(seq 1 1 $n)
do
eval echo \$x$k
done
```



5)To implement Binary search

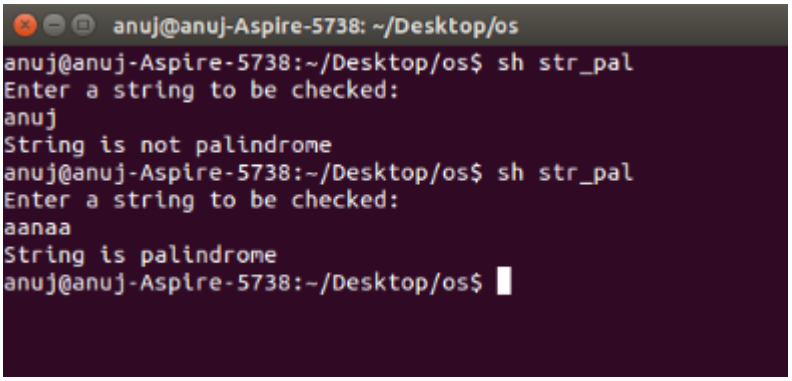
```
echo "enter no. of elements"
read m
echo "enter the $m numbers in sorted order"
for i in $(seq 1 1 $m)
do
read x$i
done
echo "enter number to b searched in above list"
read n
low=1
high=$m
mid=`echo $low+$high|bc`
mid=`echo $mid/2|bc`
while test $low -le $high
do
eval temp=\$x$mid
if test $temp -eq $n
then
echo "position is "$mid
break
else if test $x$i -gt $n
then
high=`echo $mid+1|bc`
mid=`echo $low+$high|bc`
mid=`echo $mid/2|bc`
else
low=`echo $mid-1|bc`
mid=`echo $low+$high|bc`
mid=`echo $mid/2|bc`
fi
fi
done
if test $low -gt $high
then
echo "no. is not in the list"
fi
```


anuj@anuj-Aspire-5738: ~/Desktop/os
anuj@anuj-Aspire-5738:~$ cd Desktop
anuj@anuj-Aspire-5738:~/Desktop$ cd os
anuj@anuj-Aspire-5738:~/Desktop/os$ sh anuj_binary
enter no. of elements
6
enter the 6 numbers in sorted order
2
4
5
6
8
10
enter number to b searched in above list
9
no is not in list

To implement following programs in vi-editor

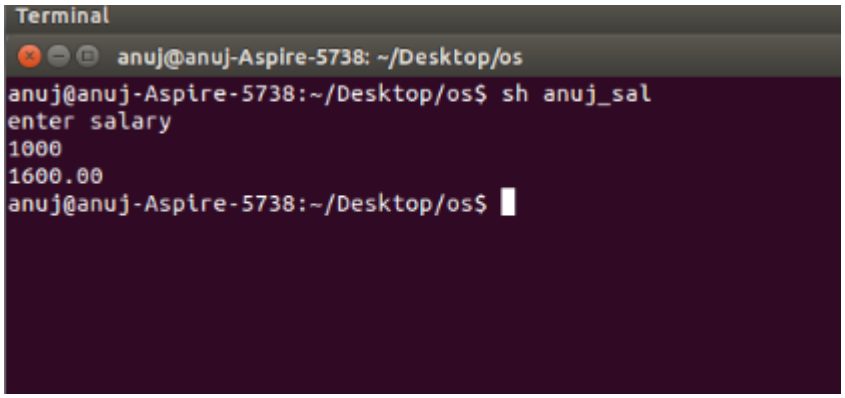
1) To input a string and tell whether it is palindrome or not

```
#!/bin/bash
clear
echo "Enter a string to be checked:"
read str
len=`echo $str | wc -c`
len=`expr $len - 1`
i=1
j=`expr $len / 2`
while [ $i -le $j ]
do
k=`echo $str | cut -c $i`
l=`echo $str | cut -c $len`
if test $k != $l
then
echo "String is not palindrome"
exit
fi
i=`expr $i + 1`
len=`expr $len - 1`
done
echo "String is palindrome"
```



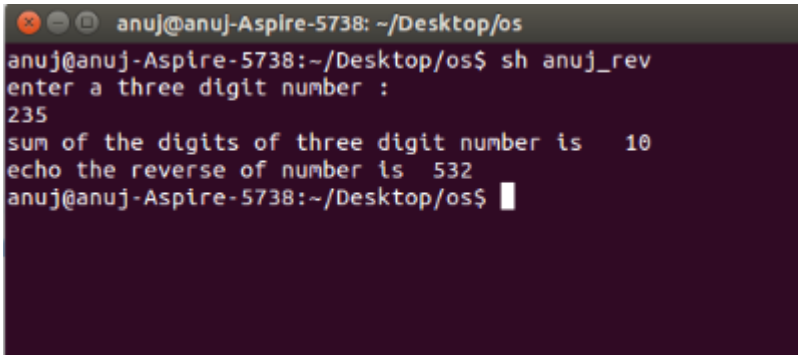
2) To find total salary if DA is 40% of basic pay and Hr is 20% of basic pay

```
echo "enter salary"
read sal
total=`echo $sal+0.40*$sal+0.20*$sal|bc`
echo $total
```



3) To input a three digit number and reverse it and find the sum of digits

```
echo "enter a three digit number : "
read num
sum=0
rem=0
rev=0
rem=`echo $num%10|bc`
sum=`echo $sum+$rem|bc`
rev=`echo $rev*10+$rem|bc`
num=`echo $num/10|bc`
rem=`echo $num%10|bc`
sum=`echo $sum+$rem|bc`
rev=`echo $rev*10+$rem|bc`
num=`echo $num/10|bc`
rem=`echo $num%10|bc`
sum=`echo $sum +$rem|bc`
rev=`echo $rev*10+$rem|bc`
num=`echo $num/10|bc`
echo "sum of the digits of three digit number is " " $sum" "
echo "the reverse of number is " "$rev""
```


anuj@anuj-Aspire-5738: ~/Desktop/os
anuj@anuj-Aspire-5738:~/Desktop/os$ sh anuj_rev
enter a three digit number :
235
sum of the digits of three digit number is   10
echo the reverse of number is 532
anuj@anuj-Aspire-5738:~/Desktop/os$

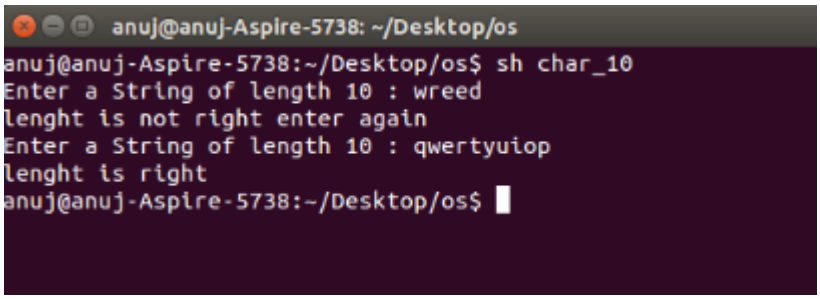
To implement following programs in vi-editor

1)Remove common files from directory

```
#!/bin/bash
# remove common files in one directory
clear
echo -en "\033[0m"
echo -en "Enter the first location  \033[5m:\e[0m "
read path1
echo -en "Enter the second location  \033[5m:\e[0m "
read path2

cd $path1
for i in *
do
  cd $path2
  for j in *
  do
    if [ $i = $j ]
    then
      rm $j
    fi
  done
  cd - > /dev/null
done
```

2) To get a string of 10 characters from user else print error message

```
#!/bin/bash
#string of length 10
i=1
while [ $i ]
do
   read -p "Enter a String of length 10 : " string
   length=`echo $string | wc -m`
   if [ $length -eq 11 ]
   then
     echo "lenght is right"
     break
   else
     echo "lenght is not right enter again"
   fi
done
```



To implement following programs in vi-editor

1) Program to rename a file

```
#!/bin/bash
# rename
# touch {1..100} ~/check
clear
echo -en "\033[0m"
echo -en "Enter the location for the file to be renamed(folder's name) \033[5m:\e[0m "
read path
cd $path
for i in *
do
  j=`echo $(basename "$i")"new"`
  mv $(basename "$i") $(basename "$j")
done
cd - >/dev/null
```

2) Program to generate backup

```
#!/bin/bash
# backup
clear
echo -en "\033[0m"
echo -en "Enter the location whose backup is to be made  \033[5m:\e[0m "
read source
echo -en "Enter the location where backup is to be made  \033[5m:\e[0m "
read path
if [ -d $path ]
then
  echo -n ""
else
mkdir --parent $path
fi
cd $source
for i in *
do
  cp $(basename "$i") $path
done
cd - > /dev/null
```

To implement following programs in vi-editor

1) To display all files in current directory to which you have read, write and execute permissions

```
#!/bin/bash
read -p "Enter the location of the folder in which files are to be searched : " path
find $path -perm 777
```

2) Write a script that accepts any number of files as input and check if it is a file or directory, if directory then report back if file then give the file names and number of lines in it.

```
#!/bin/bash
#if the given argument is file or directory

clear
for ((i=1;i<=$#;i++))
do
#  echo ${!i}
  if [ -d ${!i} ]
  then
        echo ${!i} "is a directory"
  else
        if [ -e ${!i} ]
        then
                echo -n ${!i} "is a file and has "
                echo `cat ${!i} | wc -l ` " lines "
        fi
  fi
done
```

3) Write a script that accepts any number of files as input, check if these files already exist then they are reported back, if they do not exist then create a sub directory ‘mkdir’ and copy file names into it.

```
#!/bin/bash
#if the given argument is file and it exists if yes report if no make that file under a new sub folder

clear
mkdir temp
for ((i=1;i<=$#;i++))
do
#  echo ${!i}
  if [ -e ${!i} ]
  then
        echo ${!i} "is a file "
  else
        touch temp/${!i}
  fi
done
rmdir temp 2> /dev/null
```