Linux System Administration and Support

Training Material

License

Except where otherwise noted, this work is licensed under
http://creativecommons.org/licenses/by-nc-sa/3.0/

Atlantic LINUX

Contents

Atlantic LINUX

Contents

Atlantic LINUX

Contents

Atlantic LINUX

Contents

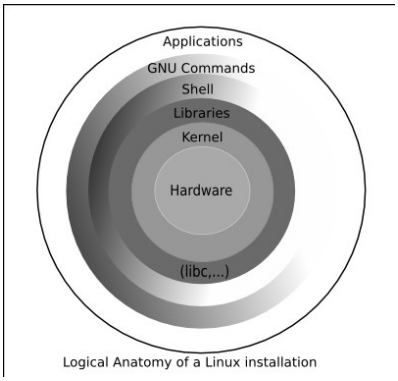
Atlantic LINUX

Contents

Introduction

What is Linux

- Kernel
  - 2.4.x
  - 2.6.x
- Distributions
  - Red Hat
  - Novell / SuSE
  - Debian
- GNU
  - compilers
  - libraries
  - editors and other tools



Logical Anatomy of a Linux installation

A brief history of Linux

Sep 1983 – Richard Stallman announces the GNU Project

Apr 1991 – Linus Torvalds announces he's working on a hobby OS

Sep 1991 – Linux Kernel 0.01

Mar 1994 – Linux Kernel 1.0 (i386)

Mar 1995 – Linux Kernel 1.2 (Alpha, Mips, Sparc)

June 1996 – Linux Kernel 2.0 (SMP, Tux the Penguin)

Jan 1999 – Linux Kernel 2.2 (64-bit, FAT32, NTFS)

Jan 2001 – Linux Kernel 2.4 (ISA PnP, PA-RISC, USB, PC Card)

Dec 2003 – Linux Kernel 2.6 (IA64, x86_64, em64t, embedded systems, NUMA)

Atlantic LINUX

From: Linus Benedict Torvalds (torvalds@klaava.Helsinki.FI)
Subject: What would you like to see most in minix?

Newsgroups: comp.os.minix
Date: 1991-08-25 23:12:08 PST

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

Linux distributions (1/2)

- Red Hat
  - Enterprise Linux (Advanced Platform, Workstation, Desktop) v5
  - Enterprise Linux (AS, ES, WS, Desktop) v4
  - Enterprise Linux (AS, ES, WS) v3
  - Fedora Core
- Debian
  - Stable (4.0 - *Etch*)
  - Testing (*Lenny*)
  - Unstable (*Sid*)
  - Derivatives (Ubuntu, Xandros, Knoppix, Progeny)

Red Hat (http://www.redhat.com/)

RHEL v5 – release March 2007

RHEL v4 - release Feburary 2005 includes 2.6 kernel, ext3 qand lvm updates, security enhanced linux compiler updates, gnome 2.8

RHEL v3 - introduced in September 2003

Enterprise Linux software has much longer support cycle (7 years)

Fedora is the proving ground for RHEL releases (effectively RHEL beta), available for free use but not supported.

Debian (http://www.debian.org/)

The debian distributions are non-commerical. a volunteer project consisting of about 1000 active developers packaging others peoples software and integrating it into a new distribution. Debian makes a release every few years which they label stable. Debians key feature is that it can be installed and upgraded over the network with a very powerful tool for managing dependencies. It is thus possible to track the current versions of Debian in development (testing and unstable).

Debian is used as a base for a number of other Linux Distributions (both commercial and non-commercial). The most notable commercial ones are Ubuntu (http://www.ubuntu.com/) and Xandros (http://www.xandros.com/). The Knoppix distribution used for system recovery, which runs directly from the CD is also derived from Debian.

Linux distributions (2/2)

- Novell / SuSE
  - SuSE Linux Enterprise Server 10
  - Novell Linux Enterprise Desktop 10 (formerly Novell Linux Desktop)
  - openSuSE 10.2
- Others
  - Gentoo
  - Mandriva
  - Ubuntu
  - Sun Java Desktop System
  - Slackware
  - Turbolinux

SuSE / Novell (http://www.novell.com/linux/suse/index.html)

SuSE was taken over by Novell in 2003. The SuSE brand is still used for some distributions. Novell branded Linux distributions consist of the core SuSE distribution and some additional components or branding.

SuSE has recently introduced OpenSuSE which is intended to be a similar project to Red Hat's Fedora being a volunteer-driven testing ground for future releases of SuSE / Novell Linux.

Others

There are a huge number of Linux distributions both commercial and non-commercial intended for a myriad of uses ranging from desktop to server to system recovery or security. http://lwn.net/Distributions/ lists 300 active distributions with a range of purposes, languages and target platforms.

Who is using Linux?

- Dot coms
  - Google
  - Amazon
  - Paypal
- Financial
  - Irish Stock Exchange
  - First Trust Corporation
  - Central Bank of India
- Entertainment
  - Ticketmaster
  - Pixar
  - Industrial Light and Magic

The Google Linux Cluster
http://www.researchchannel.org/program/displayevent.asp?rid=1680

Linux slashes costs for bank giant
http://www.computerweekly.com/articles/article.asp?liArticleID=135436

Pixar switches from Sun to Intel
http://news.com.com/2100-1001-983898.html

Red Hat Success Stories
http://www.redhat.com/solutions/info/casestudies/

SuSE Success Stories
http://www.novell.com/success/

Netcraft Web Server Survey
http://news.netcraft.com/archives/web_server_survey.html

Who's using Debian?
http://www.debian.org/users/

What is system administration?

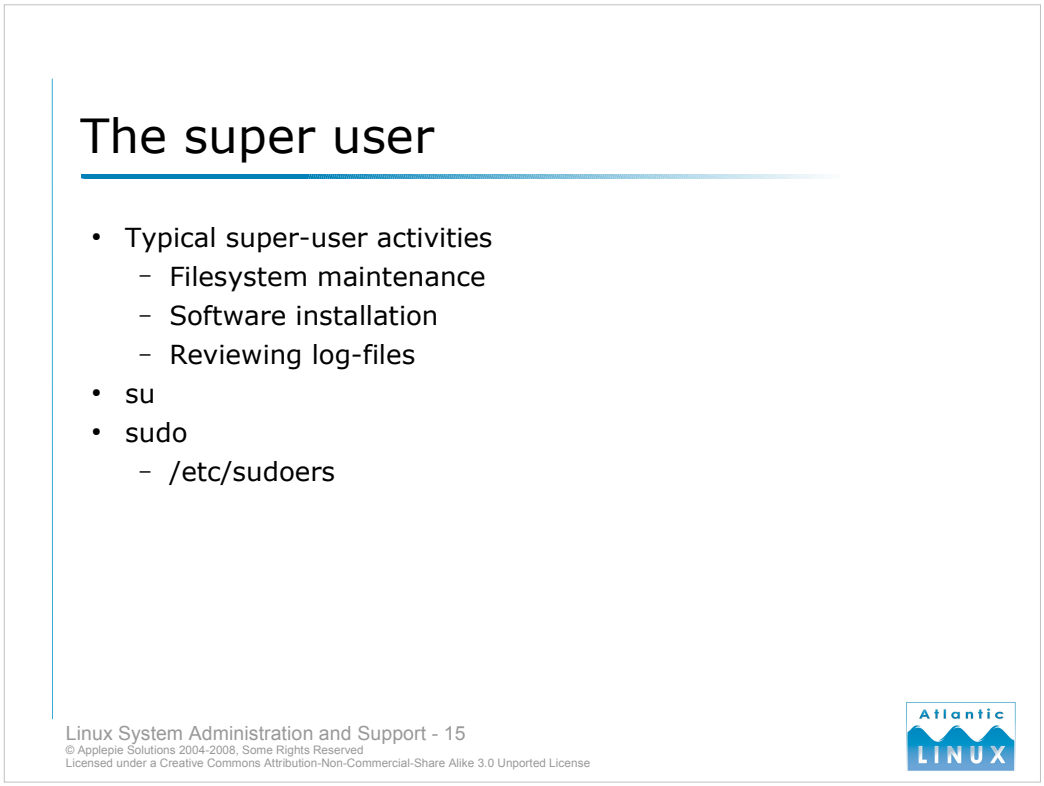
- Installation of the operating system.
- Installation of new hardware and software on the system.
- Maintaining the system with updates and patches.
- Configuring the system.
- Adding and removing users.
- Performing backups.
- Securing the system.
- Documenting the configuration of the system.
- Troubleshooting system problems.
- Performance tuning the system.
- Providing support to system users.

System administration consists of a wide variety of responsibilities, ultimately ensuring that the system continues to be able for productive use by the users. Administration starts with the initial planning and installation of the system including making decisions on what hardware and software will be required on the system. It also involves sizing and organising system storage.

After installing the operating system, the administrator will need to periodically update the system with any software updates and operating system patches which are required for operational or security reasons.

Management of users includes adding new users, removing old users and changing passwords for users. An administrator will typically provide some level of support to system users, possibly with the assistance of dedicated support staff.

The administrator is typically also responsible for backups of the system, ensuring that system is secure and that it is performing at acceptable levels. In larger organisations, some of these roles may be served by dedicated staff.



Linux uses a dedicated user account for system administration – this user is usually known as **root** and has a user id (UID) of 0. The root account usually has access to change anything on the system. You should only use the root account when it is absolutely necessary, not for day to day activities. Be especially careful with the root account and the `rm -rf` command!

When performing root activities, you can either connect by **ssh** to the system as the root user or you can use the **su** command to temporarily assume the privileges of the root user. e.g.

```
auser> su
Password: *******
root>
```

The su command can actually be used to change identity to any other system user and can sometimes be useful for verifying the configuration of another system account (particularly when invoked as `su - <user>` which replaces your current environment with the user's environment).

An alternative to giving users full access to root privileges is the use of the **sudo** command. This allows a system to be configured to allow certain users to run certain commands as root without having full access. The users and commands they can run are described in the **/etc/sudoers** file. To invoke a command with root privileges a users calls sudo command. sudo also logs a lot of information about successful and unsuccessful attempts to invoke it providing a useful audit trail of root activities (see section detailing system logs). Using sudo also prevents lots of users from knowing the root password.

System documentation (1/2)

- Distribution-specific documentation
  - Administration and install guides
  - User manuals
  - Reference guides
  - Release Notes
- man pages and the man command
  - *man [section] <page>*
  - *man -k <topic>*
  - *man -f <topic>* or *apropos <topic>*
  - *man man!*

Linux and UNIX systems in general provide a lot of online documentation in electronic formats.

All distributions are supplied with high quality manuals and reference documentation.

http://www.redhat.com/docs/manuals/enterprise/ provides guides for installation on a number of platforms, a system administration guide, a general reference guide and a number of security documents and release notes for each release of Red Hat Enterprise Linux.

Novell provides documentation for SuSE at *http://www.novell.com/documentation/suse.html* and documentation for Novell Desktop Linux at *http://www.novell.com/documentation/nld/* including a quickstart guide, a deployment guide and guides for various components of the system including KDE and GNOME.

Debian provide a full set of documentation including an installation guide, user manual and reference at *http://www.debian.org/doc/*.

The original approach to providing information on UNIX systems was the **man** command. man provides access to the information on commands, system calls, special files, file formats and others. The man pages tend to contain lots of good information but are not always very user-friendly. The man pages are split up into sections

1 Executable programs or shell commands
2 System calls (functions provided by the kernel)
3 Library calls (functions within system libraries)
4 Special files (usually found in /dev)
5 File formats and conventions eg /etc/passwd
6 Games
7 Macro packages and conventions eg man(7), groff(7).
8 System administration commands (usually only for root)
9 Kernel routines [Non standard]

System documentation (2/2)

- GNU info
- <command> -h, --help
- The Linux Documentation Project
  - http://www.tldp.org/
  - FAQs, HOWTOs, Guides
- Project specific documentation for Samba, Apache and others.

The GNU project distribute most of their manuals and reference pages in the info format. The standard tool for reading info files is the **info** command which isn't very user-friendly. The info pages tend to contain a lot of good information so it is worth getting familiar with the info command (or trying one of the alternatives such as **tkinfo**).

Most commands have some basic help built-in which can be accessed by invoking the command with either `-h` or `-help`. If in doubt, try both.

The Linux Documentation Project is a volunteer driven project found on the web at *http://ww.tldp.org/*. The project has assembled a large collection of high quality **HOWTOs**, **Guides** and **FAQs**.

Finally, a number of the larger applications used on Linux including Samba, the Apache webserver and Apache Tomcat have their own websites dedicated to providing documentation and support for these applications.

Samba
http://www.samba.org/samba/docs/

Apache Webserver
http://httpd.apache.org/docs-project/

Apache Tomcat
http://jakarta.apache.org/tomcat/

Editing system Files - introduction

- Typical editors on Linux systems
  - vi / vim
  - emacs / xemacs
- Starting vi
- Buffers
- vi modes
  - command mode
  - insert mode
  - switching modes
- Quitting vi

The **vi** text editor can be found on practically all Linux systems. There are more user-friendly and arguably more powerful text editors but anyone using Linux should be familiar with at least the basics of using vi. There are a wide range of vi-like editors all of which have the same basic characteristics.

To start using vi, invoke it at the command line as `vi filename`. If *filename* doesn't exist, vi will create it.

When you start vi, a copy of the file you are editing is placed into a **buffer**. The buffer is not written back to your file until you explicitly tell vi (details below).

The vi editor uses different **modes** to allow the user to perform different tasks. When you start the vi editor, it is in **command mode**. This is used to issue commands to the editor, move around the file, load and save files, quit and so on. To begin inserting text you must switch to **insert mode** by pressing "i". Most vi clones will display *–INSERT--* at the bottom of the screen to indicate the new mode.

When in insert mode, any keystrokes input are placed in the file. To switch back to command mode at any time press the *ESC* key.

Most commands in command mode start with ":". There are various options for exiting depending on whether you wish to save what you've input so far,

```
:q – quit vi
:q! - quit vi without prompting to save changes
:wq or ZZ or :x – save changes and quit
```

Editing system Files - navigation in vi

- Traditional navigation
- Cursor keys
- Paging
- Advanced navigation
  - start of line / end of line
  - next word
  - previous word
  - start of file / end of file
  - specifying particular lines

Navigation around a file is achieved while in command mode.

Traditionally vi has used the **h**,**j**,**k** and **l** keys to navigate around a file as follows:
h moves left one character
j moves down one character
k moves up one character
l moves right one character

Most newer versions of vi also support the cursor (arrow) keys for navigation.

You can also move through a file a page at a time using the traditional commands:
CTRL-F – move forward one page
CTRL-B – move back one page
CTRL-D move forward (down) one half page
CTRL-U move back (up) one half page

Page up and page down are supported on most modern vi clones.

There are various advanced navigation commands in vi also including:
0 – move to the start of the current line
$ - move to the end of the current line
w – move to the start of the next word
b – move to the start of the previous word
:0 – move to the first line of the file
:$ move to the last line of the file

To move to a specific line in the file, you use ***:n*** where n is the line number you want to jump to.

Editing system Files – cut and paste in vi

- Copy (yank)
  - single lines
  - multiple lines
- Paste
- Cut (delete)
  - lines
  - characters
- Inserting files

Cutting and pasting in vi is normal done in terms of lines. You can mark blocks of text within lines for cutting and pasting but it is normally easier to cut and paste the line(s). All text manipulating is done in command mode.

To **copy** (or **yank** in vi terminology) a line you use **yy**. Multiple lines can be copied by putting the number of lines before the yy e.g. **5yy** will copy the current line the cursor is on and the 4 that follow.

These lines can be **paste**d to the cursors current location using **p**.

Cutting (or deleting) is done with the **dd** command. Specifying a number before the dd again causes it to process multiple lines.

Deletion can be performed at a character level using the **x** command. Specifying a number before the x again causes it to process multiple characters.

Any deleted characters or lines can be pasted back to the cursors current position using the p command.

To insert the contents of another file into the current buffer, use the **:r *filename*** command where *filename* is the file whose contents you want to insert.

Editing system Files - finding text in vi

- Simple search
  - repeating
  - backwards
- Simple find and replace
- Advanced search
- Regular expressions

To perform a simple text search of the current buffer use the **/*phrase*** command where *phrase* is the string you want to search for. vi will automatically jump to the next occurrence of that string. To continue searching from that point for the same string, press / again.

The search can also be performed backwards in the file by using the **?** command instead of / (**?*phrase*** searches backwards for *phrase* and **?** on its own repeats the search for the previous phrase).

You can also do a **search and replace** on the contents of a file using the **:s** command. The syntax is a little more complex:

:s/*string to find*/*string to replace it with*/ (does a search and replace on the current line)

```
e.g.
:s/foo/blah/
```

:1,$ s/string to find/string to replace it with/ (does a search and replace from start to end of file)

```
e.g.
:1,$ s/foo/blah/
```

:s/*string to find*/*string to replace it with*/ (does a search and replace on the current line)

will replace each occurrence of the string “foo” with the string “blah”. The simplest form of the search and replace command only replaces the first occurrence of the string on each line it finds. To get it to replace all occurrences of the string on each line requires a slight change to the command,

:s/*string to find*/*string to replace it with*/g

Advanced searches can be performed by restricting the search to particular lines of the file.

Note also that both search and search and replace can use **regular expressions** for more complex matches.

Editing system files - advanced vi

- Undo
- Starting editing on a particular line
- Replace mode

You can **undo** the previous command using **u** or **:u**. Some versions of vi allow multiple undos while others only allow the most recent change to be undone. An undone command can be redone with **CTRL-R**.

When editing a file with vi, you can automatically start on a particular line using the following syntax

```
vi +n filename
```

where *n* is the line number you want to start editing on.

Vi includes an alternative to **insert mode** in the **replace mode** accessed from the **command mode** with the **R** command. In this mode, any text input overwrites the existing text at the cursor.

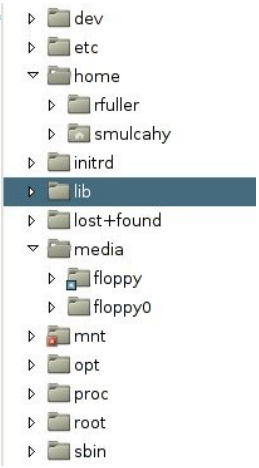
Exercise 1 - Editing

1) Create a text file containing a few sentences of text.
2) Copy a few lines of the file.
3) Use search and replace to replace all occurrences of the word “the” in your file with the string “xxxx”.
4) Copy /etc/passwd to a file in /var/tmp and practice navigating around it.
5) Remove the password for a user in the copied password file.

The Filesystem

Linux Files

- Everything is a file!
  - Docs, pictures, executables
  - Directories
  - Devices
  - Kernel internals
  - .dot files
  - hard links
  - soft (symbolic) links
- Filesystem Hierarchy Standard (FHS)
- Hard and soft links



On a Linux system, everything is a file. This includes the usual files such as documents, images, and commands. It also includes *devices*, *kernel internals* and *system settings*. Representing everything as a file is a programming abstraction which allows users to manipulate various parts of the system using a standard interface.

The filesystem tree is organised in a standard way known as the **Filesystem Hierarchy Standard** (**FHS**) which allows users of any Linux distribution (or developers of applications for Linux systems) to always find the same kinds of files in the same location.

A file's **inode** (the data structure on the disk that represents the file) can have more than one filename associated with it (a filename is just an entry in a directory data structure). The inode for the file keeps a count of how many filenames are associated with it (the link count). You can add additional filenames pointing to an inode by **hard linking** the new filename against the original filename. This increases the link count of the file. The **ln** command is used to link, passing the <*original filename*> and the <*new filename*> as arguments. A file is not deleted from the filesystem until its link count is 0.

e.g.

```
ln original_file.txt new_file.txt
```

You can also create **soft links** or **symbolic links** to files. These are more like **shortcuts** in the Windows operating system, they are aliases to the original files. You can create with them with **ln -s**. The original file can still be deleted when it has one or more soft links pointing to it.

e.g.

```
ln -s original_file.txt softlink_to_file.txt
```

File Hierarchy Standard - /

- /
- /bin
- /boot
- /dev
- /etc
- /home
- /lib

The current version of the FHS is 2.3 (29-Jan-2004). It is an evolving standard developed by the Linux community to address the needs of users, programmers, system administrators and software vendors. It describes the key features of the filesystem including where specific types of files should be found and the purpose of various system directories and sub-directories. The FHS is a component of a larger set of standards – the Linux Standard Base (LSB). The following contains some excerpts from FHS 2.3

/ - *The contents of the root filesystem must be adequate to boot, restore, recover, and/or repair the system.*

/bin - *contains commands that are required when no other filesystem is available. This can include commands used by system administrators and users.*

/boot - *contains files used in the boot process.*

/dev – *contains special device files.*

/etc - *contains configuration files. A configuration file is a local file used to control the operation of a command or system program.*

/home – *An optional location for users home directories. Alternatives like **/usr/users** are also permitted by the standard.*

/lib - *contains those shared libraries used to boot the system and run the commands in the root filesystem (the commands in /bin and /sbin).*

File Hierarchy Standard - /

- /mnt (and the mount command)
- /opt
- /root
- /sbin
- /tmp
- /usr
- /var

/mnt – *this directory is used as a temporary location for mounting other filesystems*.

/opt – *intended to be used for the installation of add-on application software packages. Applications are typically installed in a subdirectory named after the application.*

/root – *this is the recommended default location for user root's home directory*.

/sbin – *the location of commands used for system administration. This directory contains commands and other files essential for booting, restoring, recovering and/or repairing the system.*

/tmp – *this directory is used by programs which require somewhere to create (typically small) temporary files.*

/usr – *the /usr hierarchy of directories is intended as a store of **shareable**, **read-only data**. Any information which is specific to a particular system or changes over time is stored elsewhere.*

/var – *the /var hierarchy of directories is intended as a store of **writable** data. Files that are written to during normal system operation are usually stored under here rather than under /usr. Some files under /var are **shareable**, some **non-shareable.***

File Hierarchy Standard - /usr

- /usr/bin
- /usr/include
- /usr/lib
- /usr/local
- /usr/sbin
- /usr/share
- /usr/src

Atlantic LINUX

/usr/bin – *the main location for user commands.*

/usr/include – *the location of the system include files for the C programming language.*

/usr/lib – *the main location for system libraries, object files and some system commands that aren't called directly by users or scripts.*

/usr/local – *a hierarchy used for software and data installed only on this system or used by a small group of systems. Distribution software is never installed into this hierarchy so it is a safe location in which to install add-ons such as the JDK (safe in the sense that subsequent upgrades to the distribution will not overwrite or erase software installed under /usr/local).*

/usr/sbin – *this directory contains any system commands that aren't needed for system booting, repair, recovery or restore. Only commands used exclusively by the system administrator should be installed here.*

/usr/share – *this directory is intended to store read-only, architecture independent data files. This directory could be shared between multiple systems running the same OS version. /usr/share is not intended to be shared between different versions of the same OS or different OSs.*

/usr/src – *this is an optional directory in which to place read-only copies of source code. The Linux kernel source code is typically installed here by Linux distributions.*

File Hierarchy Standard - /var

- /var/cache
- /var/lib
- /var/lock
- /var/log
- /var/mail
- /var/opt
- /var/run
- /var/spool
- /var/tmp

/var/cache - *intended for use by applications to store cached data. This should only be data which the application can safely regenerate if deleted.*

/var/lib – *subdirectories under /var/lib are intended to store variable state information used by applications and the system itself. This data stored here is typically critical to the operation of a program but changes over time.*

/var/lock – *used to store system and application-specific lock files.*

/var/log – *used to store system and application log-files. The main system logs are normally stored in /var/log while other system applications store their logs in appropriate subdirectories.*

/var/mail – *this contains the system mail spool or user mailboxes where user mail is initially delivered.*

/var/opt – *this is used by packages installed in /opt when generating variable data.*

/var/run – *used to store run-time system and application data. This directory and any subdirectories usually have their contents erased at system boot.*

/var/spool – *this is a staging area for application data which the application has yet to be process. Data in this directory is usually managed by the application and will not be normally be deleted at boot time.*

/var/tmp – *this directory is intended for storing temporary files which are deleted at boot-time (as opposed to /tmp). /var/tmp should always be used for large temporary files.*

A brief look at Linux filesystems

- Simple
  - UFS
  - EXT2
- Journalling (Logging)
  - EXT3
  - ReiserFS
  - XFS
  - JFS
- Performance
  - File sizes
  - I/O Profile (read or write)

Linux filesystems can be split into simple filesystems and journalling filesystems.

Journalling filesystems are distinguished by their use of a journal to record all actions being performed on the filesystem (much like a database). Before any action is performed on the filesystem, a record of these actions is written to the filesystem journal. In the event of a system failure, the filesystem will replay any events in the journal which were not applied. The net effect of this is that journalling filesystems are more resilient and less inclined to corruption in the event of hardware or software failures.

The **Second Extended Filesystem** (**ext2**) is the most common filesystem in use on Linux systems (the original extended filesystem was used on early versions of Linux but has been entirely superceded by ext2). Due to its extensive use in Linux, it is both robust and offers good performance. Originally limited to 2GB maximum filesystem size, recent versions of the Linux kernel have extended this to 4TB. By default, 5% of the space on an ext2 volume is reserved for use by the root user. This allows the root user to easily recover from situations where the users fill up the filesystem.

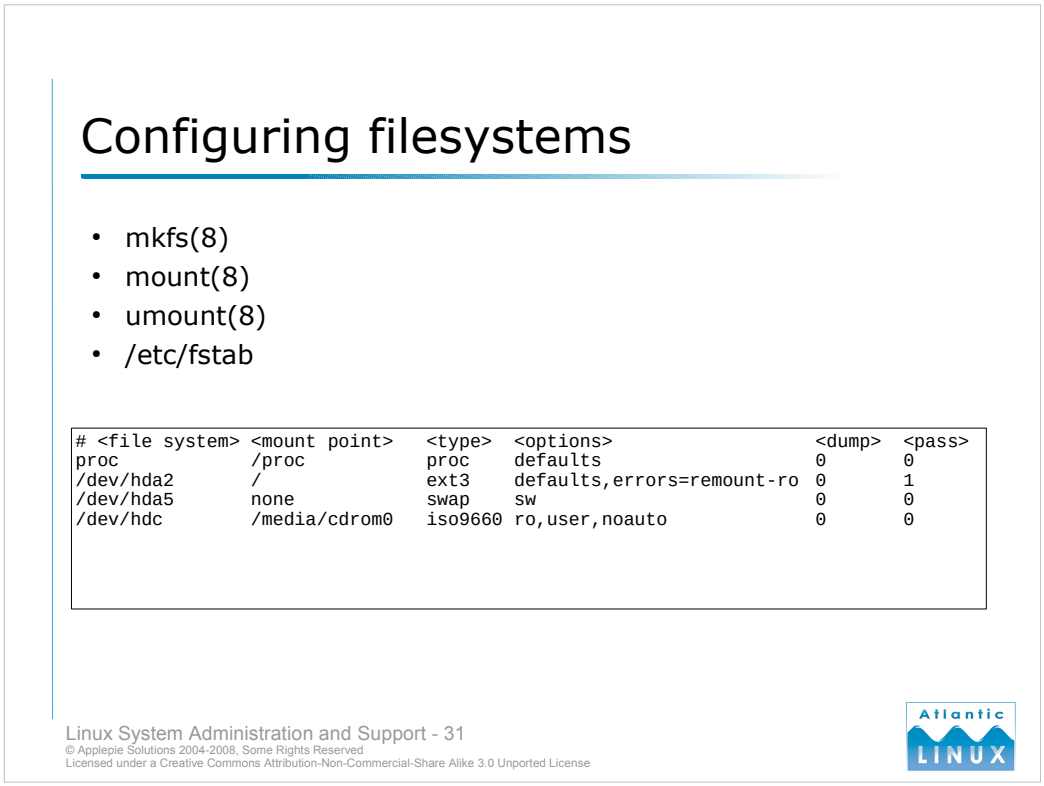
The **Third Extended Filesystem** (**ext3**) is based on ext2 but also provides journalling. It offers the same robustness and high performance of ext2 with the additional robustness of journalling. Normally, ext3 should be preferred over ext2 for new installations.

ReiserFS is a journalling filesystem which uses a variable block size. When creating ext2 or ext3 filesystems, you specify a block size to be used for files on that filesystem. All files created on the filesystem will use one or more such blocks. This can have disadvantages in the case of storing lots of small files where the files effectively much more space than they require. ReiserFS eliminates this disadvantage while providing a high performance filesystem. ReiserFS is the default filesystem on some less common distributions, e.g. Slackware and Linspire, and on older versions of SUSE/Novell Linux. ReiserFS V3 is in common production use. ReiserFS V4 has been released but is not yet integrated with the Linux kernel source.

XFS is a port of SGI's journalling filesystem to Linux.

JFS is a port of IBM's journalling filesystem to Linux.

Performance Comparisons - *http://linuxgazette.net/102/piszcz.html*



After creating a partition using a tool such as **fdisk**, **cfdisk**, **parted** or the distribution's graphical disk partitioning tool, you need to create a filesystem on the partition. The **mkfs** command can be used as follows,

```
mkfs -t <filesystem type> <partition>
```

e.g.

```
mkfs -t ext2 /dev/hdg1
```

The filesystem type is one of the supported filesystems such as *ext2*, *ext3*, *reiserfs*. Not all filesystems may be supported on a particular distribution depending on what is installed. Be very careful not to run this on a partition which already contains a filesystem!

When you have created a new filesystem on the partition, you need to make this filesystem available to the operating system. You do this using the **mount** command. The mount command is used to attach a filesystem to an existing directory in the filesystem. It is used as follows,

```
mount -t <filesystem type> <partition> <mount point>
```

The filesystem type is optional as Linux will try to automatically figure out the filesystem type for you, it can be useful when mounting non-standard filesystems, *partition* is the device name of the partition and *mount point* is a directory somewhere in your filesystem.

e.g.

```
mount -t ext3 /dev/hda1 /mnt
```

You can detach a mounted filesystem from its existing mountpoint using the umount command. The syntax is,

```
umount <mount point>
```

When a Linux system starts, it automatically mounts a number of filesystems that have already been created. To do this, it requires information about each filesystem including its partition, mount point, filesystem type and any special options. This information is contained in **/etc/fstab**. Any permanent filesystems you created should be added to this file.

Quotas (1/2)

- Specified per filesystem
  - User quotas
  - Group quotas
- Soft quotas
  - Enforced after a grace period
- Hard quotas
  - Immediately stops any writes
- Can send email notifications to users over quota

By default, a user can create as many files as they want on any filesystem that they have write permissions on. This can lead to problems for the system overall, as a Linux system normally needs to be able to write some data to the filesystem during normal operation.

As mentioned earlier, ext2 filesystems reserve 5% of the filesystem for root use by default. This goes some way to preventing individuals users filling up a filesystem. Their actions can still interfere with other users of the system.

There are a number of strategies which you can use to manage this problem. You could create a separate filesystem for each user. This would quickly become a management burden as you added and removed users.

A more feasible alternative is quotas. Quotas allow you to restrict how much space each user or group can use on a per filesystem basis.

On Novell Desktop Linux or SuSE, you may need to install the quota software package. The Linux kernel also requires support for the Quota system, the standard SuSE kernels provide this support.

To enable user quotas, you need to add the **usrquota** option to the mount options for each filesystem you want to use user quotas on. Similarly, to enable group quotas, you need to add the **grpquota** option to the mount options for each filesystem you want to use group quotas on. For example,

```
/dev/hda3       /home ext3  defaults,usrquota,grpquota    0     1
```

You then need to remount this filesystem to apply the the new flags. The mount command has a special option, remount for doing this. For example,

```
mount -o remount /home
```

Quotas (2/2)

- Quota commands
  - quotacheck
  - edquota
  - warnquota

```
Disk quotas for user smulcahy (uid 1000):
  Filesystem                  blocks       soft       hard     inodes     soft     hard
  /dev/sda2                     1230       2000       3000        553        0        0
```


You now need to create the metadata files used by the quota system. This is achieved using the **quotacheck** command as follows,

```
quotacheck -m -a -u -g
```

You are now ready to set quotas for users and groups. The command for creating quotas is **edquota**. To create a user quota, you invoke it as,

```
edquota -u <username>
```

To create a group quota, you invoke it as,

```
edquota -g <group>
```

Users have a **soft limit** and a **hard limit** on each filesystem. When a user reaches their soft limit, they have a grace period before they can no longer write to the file system. When the user exceeds the grace period or the hard limit, they will no longer be able to write to the filesystem. The grace period can be specified per filesystem using **edquota** with the **-t** option.

The **warnquota** command can be run periodically to send an email warning to users over quota. It would normally be run as scheduled job (see section on scheduling jobs).

File permissions and ownership (1/2)

- Setting permissions
  - Symbolic mode

    chmod <groups> <add/remove/set> <permissions> <file>

    e.g.

    chmod u=rwx,g=rx,o=rx foo

    chmod u=r,g=r,o= foo

  - Octal mode

    chmod <mode> <file>

    e.g.

    chmod 0755 foo

    chmod 0440 foo

The **chmod** command is used to change file permissions. It can be used in two different ways - **symbolic mode** or **octal mode**.

Using chmod in symbolic mode, it is invoked with the following format

```
chmod <groups> <add/remove/set> <permissions> <file>
```

groups is one or more of **u**,**g**,**o** and **a** where

- **u** is the user that owns the file
- **g** is other users in the file's group
- **o** is other users not in the file's group
- **a** is all users.

Permissions are added using the + operator, removed using the – operator or set (over-riding any existing permissions) using the = operator.

Permissions consists of one or more of the letters **rwxXstugo** for **read** (**r**), **write** (**w**), **execute** (**x**), **execute only if the file is a directory or already has execute permission for some user** (**X**), **set user or group ID on execution** (**s**), **sticky** (**t**), **the permissions granted to the owner of the file** (**u**), **the permissions granted to the group of the file** (**g**) or **the permissions granted to others** (**o**).

octal mode is a more powerful but less intuitive way of using chmod. Using chmod in octal mode, it is invoked with the following format

```
chmod <octal mode> <file>
```

octal mode is a series of 1-4 octal digits with values of 4, 2, 1 or 0. Missing digits are treated as 0. Multiple permissions are set at the same time by adding the values together. The first digit selects the **set user ID** (4), **set group ID** (2) and **stick** (1). The second digit selects permissions for the owner – **read** (4), **write** (2) and **execute** (1). The third digit selects permissions for the group (with the same values as the owner digit) and the fourth digit selects permissions for others.

File permissions and ownership (2/2)

- Changing file ownership
  - chown
- Changing file group
  - chgrp

- Special permissions
  - The sticky bit
  - The setuid/setgid bit
  - File owner

File ownership is managed by 2 commands, **chown** for changing the **file owner** and **chgrp** for changing the file group.

chown and **chgrp** have a similar syntax. The are called as

<command name> <owner or group> <file>

for example,

```
chown smulcahy foo

chgrp users foo
```

In addition to the normal read, write and execute permissions you can apply to files, there are 2 special permission bits which can be applied to files.

The **sticky bit** normally only has an affect when applied to directories. When it is set on a directory, any files in that directory can only be renamed or deleted by their owner (or the super-user). Without the sticky bit, any user with write permissions on a directory can rename or delete files in that directory. The sticky bit is useful for world-writable directories such as **/tmp** and **/var/tmp**.

There is a facility in Linux systems where a running command can make a system call to change either the user it is running as (**setuid()**) or the group it is running as (**setgid()**). Only the super-user can make these system calls. The operating system also allows files with the **setuid** or **setgid** bits set to run as their owner/group regardless of who invokes them. See the security section for more details of setuid and setgid..

Monitoring free space and inodes

- Files and directories
  - du
- Mounted filesystems
  - df
- -h option improves readability

The **du** command summarises the space usage of each file specified as an argument to it. It can used to check the space usage of both files and directories (it defaults to analysing directories). Usage is reported in kilobytes by default, the **-h** option makes the output for larger files more readable.

The **df** command reports filesystem disk usage. By default, df lists the space usage on all mounted filesystems. The **-h** is also supported by df.

Filesystem maintenance

- Tuning
  - tune2fs and reiserfstune
- Repair
  - fsck and reiserfsck
  - debugfs and debugreiserfs
- Filesystem identifiers
  - device names
  - labels
  - UUIDs
- Resizing
  1. partitions
  2. filesystems

Atlantic LINUX

Tuning involves changing various parameters of the filesystem.

The **tune2fs** command is used to tune ext2 and ext3 filesystems. You can use it to alter the following parameters,
- alter the interval between automatic filesystem sanity checks
- set the behaviour of Linux kernel when a filesystem error is encountered
- set the size of the reserved area of a filesystem and assign a user and group to this area.
- add an ext3 journal to an ext2 filesystem and control journal size and device used for journalling
- set a human-readable label or generate a universally unique identifier (UUID) for the volume

The reiserfstune command is can be used to alter the following parameters on ReiserFS filesystems,
- journal parameters (size, transaction size and location)
- label and/or UUID

In the event of a filesystem error, there are a number of tools which can be used to identify and possibly repair the error. The standard tool is **fsck** which is actually a wrapper for a number of filesystem-specific repair tools (**fsck.ext2**, **fsck.ext3** and so on). fsck is normally run against mounted filesystems periodically at boot-time. In the event of an error being found it will either prompt to repair it (if the error is low severity) or recommend rebooting to maintenance mode for more detailed analysis. ReiserFS provides its own tool - **reiserfsck** which serves the same purpose, performing checks and optionally repairing filesystem errors encountered. There are also low-level tools (**debugfs** and **debugreiserfs**) supplied with most distributions. These are for low-level debugging and analysis of the filesystem contents and are rarely used for day to day maintenance.

Filesystems can be identified either by their device name (more on these in a later section), a human-readable label assigned using the appropriate tuning tool or a universally unique identifier (UUID). The disadvantage of using device names is that they are tied to the devices physical location in the system. Labels and UUIDs are not system dependent and will work equally well if the device is moved within a PC or moved to another PC.

Note that ext2, ext3 and reiserfs filesystems can be enlarged or shrunk using the **resizefs** and **resize_reiserfs** commands. These commands do not manipulate the partition size, so in the case of enlarging filesystems it is vital to resize the partition before attempting to resize the partition itself (using a tool such as **parted**).

Exercise 2.1 - Files

1) Review a full directory listing (ls -la) for the following directories,
   - /
   - /var/log
   - /etc
2) Using the man page for the ls command, explain the contents of each column.
3) Review some files under /var/log using the less command.
4) Create a hard link and a symbolic link for a file in your home directory.
5) Create a file in your home directory containing some text.

Exercise 2.2 - Files

6) Create a symbolic link and a hard link to this file.
7) Remove the original file and explain whether each link still exists and what it points to.
8) Where in the filesystem should the following files be?
   - a configuration file
   - a large 3rd party software package
   - documentation for a standard system package
   - a log file for a system daemon
   - a file related to booting
   - a user's email

Exercise 2.3 – Files

9) **Using both symbolic mode and octal mode, set the following permissions on sample files in your home directory (see the touch command for creating test files),**

- **readable and writeable by you only**
- **readable and executable by you only**
- **readable, writeable and executable by you and readable and writeable by everyone else**

Atlantic LINUX

Exercise 2.4 - Files

10)Create a new filesystem.

11)Add an entry in /etc/fstab for the filesystem, so that its default mount point is /mnt/new

12)Enable quotas on a filesystem.

13)Create a quota for a user giving them a soft limit of 1MB and a hard limit of 1.2MB

14)Report the space used by your home directory and the free space available on your filesystems.

15)Run a filesystem check.

For the purpose of the exercise, the following commands may be used to create a "virtual" 5MB disk drive, on which a filesystem can be created (NB: be **very** careful with the *dd* command – if it is used incorrectly, it can irreversibly destroy data on your system):

dd if=/dev/zero of=~/fake_disk_drive bs=1k count=5000

losetup /dev/loop0 ~/fake_disk_drive

/dev/loop0 may now be treated as a normal disk drive, i.e. a filesystem can be created on it which can then be mounted.

Once you are finished the exercises, unmount the new filesystem, and execute the following commands to clean up:

losetup -d /dev/loop0

rm -f ~/fake_disk_drive

Booting

The boot process

BIOS → MBR
MBR → **Boot Loader**
Boot loader → **Linux kernel**
Linux kernel → **init**
init → **getty**.
getty → **login**
login → **shell**

When an x86 system initially boots, the systems **BIOS** (basic input/output system)loads the first 512 bytes from the **boot partition** on the **boot disk –** this area is known as the **MBR** (master boot record).

When an operating system is installed on a system, it usually places a piece of software called a **Boot Loader** in the MBR (or enough of the boot loader to load the rest from somewhere else). Linux uses a number of different boot loaders including **LILO** (Linux Loader) and **GRUB** (GRand Unified Boot loader).

The sole purpose of the boot loader is to load the **operating system kernel.**

The Linux kernel initialises devices on the system and passes control to the parent of all Linux processes – **init**. init always has a **process ID** (**PID**) of 1.

The init process starts a number of processes by executing various scripts listed in the **/etc/inittab** file. One of the processes started is **getty.**

Whenever something connects to the system through the **console** getty starts the **login** process (remote connections are handled via a process such as **sshd**).

After authenticating a user, login starts a **shell** for the user and passes control to that.

Boot loaders

- LILO
  - ease of installation
  - physical location of kernel
  - 1024 cylinder limit
  - /etc/lilo.conf
- GRUB
  - filesystem aware
  - interactive mode
  - broad range of operating systems
  - /boot/grub/menu.lst
- NTLDR (Windows)
  - Capable of loading other operating systems

The purpose of the boot loader is to pass control of the system to the operating system kernel. There are 2 boot loaders normally used with Linux - LILO and GRUB. These differ in their features and capabilities. Windows also uses a boot loader to to pass control to the operating system. Note that boot loaders can normally start different operating systems so the Windows boot loader can also be used to start Linux and GRUB and LILO can be used to start Windows.

LILO (the **LI**nux **LO**ader) is an established boot loader used to load both Linux and Windows operating systems. It passes control to the Linux kernel which it expects to find in a specific location on the hard disk. This means that any changes to an installed kernel require LILO to be reloaded. LILO also has some problems on older systems loading kernels which are located above the 1024th cylinder on the hard disk.

Distributions have started moving to **GRUB** (the **GR**and **U**nified **B**ootloader) which does not require kernels to be in specific locations on the hard disk and can actually read file systems. GRUB also has an interactive mode which allows you to choose different kernels to load at boot-time (LILO can only load kernels which have been preconfigured). GRUB has no 1024 cylinder limits. GRUB also has support for booting more operating systems than LILO (including *DOS*, *Windows 9x/NT/2000/XP*, *Linux*, *BSD*, *BEOS*, *SCO Unix*, *OS/2* and *Solaris*).

The Windows boot loader, **NTLDR** can also be used to load Linux operating systems on dual boot systems. It can be configured to pass control to a Linux boot loader which handles the actual loading of the Linux operating system. Full details of this configuration are available in *http://www.tldp.org/HOWTO/Linux+NT-Loader.html*

Boot options

- Passed via
  - Bootloader config file
  - Boot prompt
- General options
  - single
  - root=<device>
  - acpi=off
  - init=<program>
- Hardware-specific
  - PCI
  - SCSI, IDE
  - Ethernet

At boot-time, the kernel can be passed various commands which control the behaviour of the kernel after booting. These options are normally specified in the boot loader configuration file but can also be passed at the boot prompt.

Some standard options include **single** which tells the Linux kernel to start in *single user mode* (for maintenance); **root=<*device*>** which specifies what device the Linux kernel should use for the root filesystem and **init=<*program*>** which specifies what program the Linux kernel should pass control to after booting (normally, */sbin/init* but this can be any valid Linux command including something like */bin/sh*). There are many more boot-parameters documented in the kernel source (*Documentation/kernel-parameters.txt*) or in *http://www.tldp.org/HOWTO/BootPrompt-HOWTO.html*. One parameter which can be useful for booting systems with unreliable BIOSs is **acpi=off** which disables the *Advanced Configuration and Power Interface* subsystem in the kernel.

There are also a wide-number device-specific parameters which can be used to control the behaviour of various device drivers. These are also covered in the documentation referenced above. We will look at some specific parameters in subsequent sections relating to specific items of hardware.

Runlevels, init and rc.d

- Starting/stopping
- Adding new ones
- `/etc/init.d`
- Run-levels
  - `/etc/inittab`
  - `/etc/init.d/rcn.d`
- Distribution differences
  - SuSE / Novell Linux Desktop
  - Red Hat
  - Debian

When init starts, it starts any system commands that will be run in the background – these are called **daemons** or **services** and include commands like the system logger (**syslogd**), the internet super-server (**inetd**), the ssh service (**sshd**) and others. These are usually started with by a collection scripts in */etc/init.d*.

The scripts in **/etc/init.d** can be used to start, stop and check the current status of these services. New scripts can also be added in here if the administrator wishes to manually add new services to the system.

A Linux system can be started in various configurations. For example, if a system needs maintenance, it can be run in **single user mode** which prevents any other users from logging in and stops all non-essential services. Similarly, a system can be started in a multi-user configuration with no network services like webservers running.

These different configurations are specified by using what Linux calls **run-levels**. Linux has a number of standard run-levels (1 is *single-user*, 3 is *full multi-user text mode*, 5 is *full multi-user graphical mode*) and the system can be switched between the run-levels with the **telinit** command. The default run-level is defined in */etc/inittab*

Linux systems contain a series of directories under */etc* or */etc/rc.d* called *rc0.d*, *rc1.d*, *rc2.d* and so on. Each of the directories contains **symbolic** links to the scripts in */etc/init.d* specifying what services should be started and stopped for each run-level and specifying what order these services should be started and stopped. SuSE and Red Hat provides the **chkconfig** tool for managing these or they can be modified manually. Scripts start with either “S” or “K” followed by a number. “S” scripts start a service, “K” scripts stop a service. The scripts are executed in increasing numeric order.

There are subtle differences between the run-levels on different distributions. You should install the *sysv-rc* package on Debian to product the same behaviour as SuSE and Red Hat. Note that Debian uses */etc/rcN.d* while SuSE uses */etc/init.d/rcN.d*. Apart from this, the behaviour is the same between distributions.

Shutdown and rebooting

- shutdown
  - -r
  - -h
  - time
    - now
    - hh:mm
    - +m
- reboot
- halt
- warm vs. cold reboots

The **shutdown** command is used to shutdown a Linux system in a controlled, secure fashion.

The command takes a number of options including a mandatory time argument which specifies when the shutdown takes place. The shutdown command sends a message to all users that the system is shutting now and sends the TERM signal to all system processes giving them the chance to shut themselves down in a controlled manner. The shutdown command can either be used to **halt** the system (with the **-h** option) or **restart** the system (with the **-r** option). On systems with a modern power management system (and the appropriate power management options compiled in the system kernel, halt will also power off the system).

The **reboot** and **halt** commands are alternative commands (which normally invoke the shutdown command).

A Linux system can either do a warm boot or a cold boot. This behaviour is controlled by a kernel setting which can be over-ridden with a boot flag. A cold boot is one where the system performs a full reset, the BIOS conducts a memory test and any other diagnostics normally performed at power-on. A warm boot avoids a full reset and such diagnostic tests and usually results in a faster boot time. The default boot mode was changed during the 2.0 kernel timeframe from warm to cold as some older systems fail to reboot properly during a warm boot.

Dual boot configurations

- Overview
- Not limited to 2 operating systems
- Windows on primary partition
- Issues
  - 1024 cylinder limit
  - MBR reset
  - Kernel changes
- Advanced
  - Partition hiding
  - Partition re-ordering

Introduction

It is possible to have both Windows and Linux operating systems installed on a system at the same time. You can only run one operating system at a time on the system but you can switch between the installed operating systems at boot-time. All modern boot-loaders can be configured to present a menu of installed operating systems to choose from. The normal configuration is to have one instance of Windows and one instance of Linux installed on a system but it is possible to have many more operating systems installed and available at boot-time (different versions of Windows and different distributions of Linux for example).

Procedure

It is normally recommended to install Windows on the system first as some versions of Windows require that they be installed on the first system partition (or that the first system partition be an NTFS or VFAT filesystem). Note that LILO and GRUB both have mechanisms for hiding partitions or re-ordering partitions at boot time. When installing Windows, make sure to leave some free space on the drive for your Linux partitions (unless you are using a separate disk for the Linux installation). After completing the Windows installation, boot from your Linux install media and ensure that you use the free space rather than installing over the existing operating system (modern Linux installers will automatically recognise the other operating system and default to providing a dual boot installation). The installer will usually automatically configure the boot loader to provide both operating systems as boot options. You may wish to change the default operating system to the one you normally require. The boot loader will be installed to the MBR on the disk. The boot loader will pass control to the Windows boot loader (at the start of the Windows partition) if Windows is selected. In the case of Linux being selected, the boot loader will load the Linux kernel directly.

Issues

1. If you install/re-install Windows after installing Linux, it will normally overwrite the bootloader on the MBR with a Windows boot loader. You will need to reboot using the Linux distribution rescue disk and manually reinstall the bootloader.
2. If using the LILO boot-loader, any changes to the installed Linux kernel require a reinstall of LILO before rebooting the system.
3. Some older PCs have a BIOS limitation which prevents the boot loader loading any data beyond the 1024th cylinder on the disk. It is thus recommended that files required for booting are placed on a small partition (/boot) which occupies an area below cylinder 1024. This is not a consideration on modern systems.

Exercise 3 - Booting

1. Review the ps man page and run the ps command with options showing the hierarchy of processes.
2. Change the default runlevel on your system to the non-graphical mode and reboot the system.
3. Add an option to the boot loader to boot the operating system in single user mode.
4. Add a new service to be started up in your default runlevel (either manually or using *chkconfig* on Redhat/ SuSE, *update-rc.d* on Debian)
5. Disable the service again.
6. Perform a shutdown of the system at a specific time in the future using either time format.

Adding hardware

Useful commands

- Diagnostic commands
  - dmesg
  - lspci
  - lsusb
  - hwinfo
  - dmidecode
- Loading kernel modules
  - modprobe
  - hotplug

Most modern hardware should install on a modern PC running Linux without any problems. When problems do occur, it is useful to be able to retrieve as much diagnostic information as possible from the running Linux system. There are a number of sources of diagnostic information on any Linux system.

dmesg

The dmesg command prints the contents of the *kernel ring-buffer*, an area in memory to which the kernel writes status messages. At system startup, it contains the boot messages and little else. Over time, it also accumulates other messages from the kernel. Most distributions put a copy of the boot-time messages into a file in */var/log* (the filename differs between distributions with Debian using */var/log/dmesg* and SuSE using */var/log/boot.msg*). This log can be useful for reviewing problems with detecting hardware or loading the kernel modules required by that hardware.

lspci and **lsusb**

The lspci command displays information about the PCI buses and devices attached to them. It can be used to get some basic information about devices which are installed on the system but not being properly recognised. The lsusb command displays similar information about the USB bus.

hwinfo and **dmidecode**

The hwinfo and dmidecode can be used to probe and display information about the system and attached devices. hwinfo is normally found on SuSE systems. The dmidecode command will need to be installed to be used. The output from hwinfo can also be displayed via Yast2 under SuSE and NLD. Information available include details of attached devices and system details such as the system model, serial number and processor features (speed, description, processor type and processor features such as hyperthreading).

Kernel modules

Any devices you want to use under Linux require a driver in the kernel. Kernel drivers can either be compiled into the kernel or can be built as modules which are loaded when an driver is required. We will cover kernel modules in more depth later on. All current Linux distributions use the hotplug system to automatically load kernel modules for recognised devices so it should not normally be neccesary to use modprobe directly (but can be useful for testing/diagnosing new hardware).

General issues

- Unrecognised devices
  - unrecognised device id
  - attempt manual unload
  - information on device chipset required
  - hardware compatibility databases
- Device firmware
  - required for proper operation of device
  - usually available from vendor or driver developer
- Resource conflicts
  - PCI (interrupts allocated by BIOS, shared interrupts ok)
  - ISA, PCMCIA (probing issues and conflicts)

Unrecognised devices

Each Linux device driver contains a list of unique device ids that it supports. The Linux hotplug system uses this list to automatically select a driver to be loaded when a new device is plugged in (either at boot-time or subsequently). If the driver does not contain your device's id, it will not be loaded automatically. It may still be possible to manually load a driver for this device. At the lowest-level, this is done using the insmod of modprobe commands to load the appropriate kernel driver module. At a higher level, most distributions provide a hardware management interface which lets you manually select a driver to load. If you are lucky, the driver will pick up the device automatically. In rare situations, you may need to provide options to the driver indicating what interrupt, i/o ports or dma settings the device is using.

Hardware Compatibility

All the leading distributors maintain some database outlining the *supported* or *compatible* hardware for their distribution which may be worth reviewing before purchasing new hardware.

- *http://cdb.suse.de/*
- *http://bugzilla.redhat.com/hwcert/*
- *https://wiki.ubuntu.com//HardwareSupport*

Device Firmware

Lots of modern hardware requires firmware to be loaded onto the device before it can function properly (this function used to be served by an EEPROM on the device, but cost-savings have led to this being replaced by cheaper RAM device). You may experience problems with devices if you have an older version of the firmware or if the firmware is unavailable. The latest firmware is usually distributed with the driver for the device but it may be useful (especially for wireless cards, to check for the latest version of the firmware on the driver's site).

Resource Conflicts

Devices in common use today include PCI, ISA and PCMCIA devices. With PCI, interrupts are allocated to devices at boot-time by the system BIOS. Interrupt conflicts are thus very rare with PCI devices. With ISA and PCMCIA devices, the subsystem and the driver probe for free IRQs and other resources. This probing can cause problems in itself - other installed devices may hang as a result of the interrupt probing process or drivers may mistakenly allocate a used IRQ. You can review existing used interrupts on a Linux system by looking at */proc/interrupts*. Note that some older devices may be limited to using certain IRQs. Note also that PCI devices often "share" IRQs without any problems.

/dev filesystem

- /dev
  - major and minor numbers
  - mknod
  - MAKEDEV
- udev
  - /dev files created on the fly
  - persistent device naming possible
- sysfs
  - kernel interface to device information

Linux systems provide an interface to devices under a special filesystem mounted at **/dev**. Traditionally, this filesystem contains lots of special files and directories which map to all possible devices that could be connected to the system. Internally, Linux uses these files to access devices - it is also possible to directly access some devices on the system via these files although more normal to use a program that understands the device in question. But something like

```
cat hello.wav > /dev/audio
```

should play the given file through the audio device provided it is a valid sound file and the audio device is working properly.

Problems accessing devices on a system might be caused by the appropriate device file not existing under /dev (although this is much less of a problem in modern distributions). During installation, the /dev directory would normally have have been populated with device files by running the special script **MAKEDEV**. There could be instances when adding new hardware where device files were missing. If the MAKEDEV did not know about the devices for instance. The **mknod** command can also be used to manually create special device files. Each device file has 2 numbers associated with it, labelled the *major* and *minor numbers*. The major number is used by the Linux kernel to identify what driver to associated with the device file, the minor number is used by the driver to differentiate between multiple devices served by that driver. For example, */dev/hda1* has a major number of 3 (for the IDE driver) and a minor number of 1 (the first partition on the first disk).

Current Linux distributions running the 2.6 kernel have moved to using 2 new technologies - **udev** and **sysfs**. With the traditional approach, /dev contained lots of device files for devices that were not installed in the system but might some day be connected to the system. udev dynamically creates device files in /dev when they are installed on the system. udev can also interact with the new sysfs filesystem provided by the 2.6 kernel which provides information about connected devices including the device type, size, manufacturer and so on. This information is retrieved directly from the kernel so can be relied on to be accurate. udev can also be used to create persistent names for devices (for example */dev/intelGigaBitNetworkCard*) although most Linux distributions aren't currently using this functionality.

/proc filesystem

- /proc/NNN/
  - one per process
- /proc/cpuinfo
  - processor information
- /proc/version
  - kernel version
- /proc/meminfo
  - memory details
- /proc/devices
  - device drivers and major numbers
- /proc/bus
  - a tree of information about system buses (pci, usb, ...)

The /proc filesystem is virtual filesystem. That is to say, it does not represent actual files, rather it is an interface into the kernels memory structures displayed in a human readable fashion. It allows you to see a lot of system information (albeit in a raw format). Most of this information can also be viewed using various tools like lspci which actually retrieve their information from the /proc filesystem and format it to be more readable.

The /proc filesystem can be very useful for diagnosing problems and retrieving information on the system that may not be easily available through other means, we will look at some of the most useful sections within it but note that there are many more subsystems available. More documentation is available in the kernel source (*Documentation/filesystems/proc.txt.gz*).

/proc/PID
/proc contains a subdirectory for each active on the system process. This subdirectory contains files describing the status of the process, the resources it is using, its shell settings and so on. Much of this information is also available from ps using appropriate switches (cat /proc/self/status).

/proc/cpuinfo
Contains a lot of information about the processor(s) on the system. Can be useful for checking if all processors have been detected or if they are being recognised correctly.

/proc/meminfo
Gives a lot of information about system memory usage (including virtual memory). The vmstat command provides most of the same information in a more readable format.

/proc/devices
Lists the loaded kernel drivers and their major numbers.

/proc/bus
This contains a tree of directories and files which describe the system buses visible to the kernel and the devices attached to each one.

/proc/interrupts, /proc/dma, /proc/ioports and /proc/iomem
These list out the various resources currently in use in a system. /proc/interrupts can be particularly useful both to see the interrupts in use on a system and the activity of devices.

Storage devices

- IDE
  - /dev/hd*
- SCSI
  - /dev/sd*
- SATA
  - /dev/sd*
- Commands
  - scsiinfo
  - lsscsi
  - hdparm

Storage devices used in modern systems include hard-drives, optical devices (cdrom and dvd-roms), usb memory sticks and flash memory (compact flash, smart media and so on). Hard-drives on low-end systems have traditionally used IDE (also known as ATA) with higher-end systems using SCSI. Currently, SATA devices are entering both ends of the market and will be the predominant device on systems within a few years. Linux automatically detects new storage devices added to the system and assigns an identifier as according to what type of device it is.

IDE

IDE device names are labelled ***/dev/hdX*** where *X* is a letter starting at **a**. So the first IDE device in a system is **/dev/hda**. The first IDE device is the master on the first physical IDE channel of the motherboard. The second device is the slave on the first physical channel and so on. IDE devices are split into partitions for normal use and the partitions are identifed numerically starting at **1**. So the first partition on the first device is identified as **/dev/hda1** and so on. Note that only some storage devices use partitions. Optical devices using the ISO9660 filesystem (the standard cdrom filesystem) are accessed without a partition number (so to mount a cdrom for use by the operating system you would use `mount /dev/hdc /mnt` assuming the cdrom drive was the master device plugged into the second IDE channel).

SCSI

SCSI device names are labelled ***/dev/sdX*** using the same conventions as IDE. The drive letters are mapped in the order of the SCSI device IDs and SCSI bus IDs (in the case of a system with multiple SCSI adapters). So device 0 on the first adapter would be **/dev/sda** and so on.

Others

SATA devices, USB storage devices and flash memory devices are usually mapped into the SCSI namespace. You can verify this using the dmesg command - the device name assigned to a particular device will be reported out when the device is connected to the system.

Commands

There are a number of commands for managing storage devices on the system. The **lsscsi** command lists out all SCSI devices (or any other devices using the SCSI namespace including SATA drives) including information about the type of device it is, its device name and its device and bus details. **scsiinfo** provides more detailed information about a particular device including device error details. The **hdparm** command is for advanced users and can be used to view and modify settings on IDE disks such as which DMA mode is in use (DMA can have a large performance impact).

Network devices

- Device names
  - /dev/ethN (/dev/trN, /dev/loN, /dev/sitN, /dev/pppN)
- Linux 2.4 device detection
- Linux 2.6 device detection
- Device ordering
  - module load order
  - PCI BIOS Ids
- Commands
  - ifconfig
  - nameif
  - mii-tool

Linux network devices have different device names depending on the driver used. Ethernet, the most common category of network devices, uses ***/dev/ethN*** where *N* is a number starting at 0. The numbers are assigned in the order that the Linux kernel loads drivers for the devices.

In Linux kernels up to 2.4, only one network device was automatically detected (adding support for additional devices involved passing arguments at the boot prompt or through the boot loader configuration file). Linux kernels from 2.6 onwards do not have this problem. In either case, it is possible that the names of your network devices may change if something happens to change the order in which your network device drivers are loaded. This can typically happen between kernel upgrades or other system changes.

With modern distributions using the 2.6 kernel and loadable modules, you can address the problem by adding lines to the **/etc/modules.conf** which ensure a specific order to loading kernel modules. If you have 2 PCI devices which both use the same driver, the order should depend on their PCI BIOS ids (which should remain the same over the lifetime of the machine unless you physically move the cards to different slots).

Finally, you can also use the **nameif** command to assign a particular device name to a particular MAC address (MAC addresses are worldwide unique IDs assigned to every ethernet card, you can view the MAC address for a device using the **ifconfig** command). This could be run at boot-time to reassign device names to arbitrary devices.

The status of a particular ethernet device and some diagnostic information can be reviewed using the ifconfig command which can be used to view network device details, transmission statistics, change settings and bring the device online or offline. **ifconfig -a** can be used to list all installed network devices.

Finally, some devices can have problems auto-negotiating speeds or duplex settings with malfunctioning switches or hubs. The **mii-tool** command lets your view and over-ride the default behvaiour of a card during this scenario.

Exercise 4.1 – Adding Hardware

1. Identify the network card installed on the system using the lspci tool.
2. List the USB devices attached to the system and identify which types of USB controllers are attached to the system.
3. Identify how many CPUs are installed in your system and what type they are (processor model, manufacturer and speed).
4. Retrieve the system model and serial number using either the hwinfo or dmidecode commands.
5. Review the contents of the /dev system and indicate whether the system is using udev or a traditional static /dev.

Exercise 4.2 – Adding Hardware

6. Identify the major and minor numbers of the cdrom device and the hard drive.
7. Using the /proc filesystem report the total amount of memory in the system, the kernel version in use and the interrupt used by the first network device.
8. What is the make and model of the first hard-drive?
9. What is the MAC address of your network device? What is its IP address?

The X Window System

Overview of X

- Introduced in 1984
- Standard environment for UNIX Windowing systems
- Network-transparent graphical windowing system
- Capabilities:
  - Network transparent
  - Graphical capability
  - Not tied to any particular display software
- Server
  - The program that talks to the keyboard, mouse and graphics hardware
- Client
  - The program requesting draw operations
- X.Org and XFree86

Officially named *The X Window System* but also known as *X* or *X11* (not X Windows!). It was developed in 1984 in a collaboration between MIT and Digital Equipment Corporation. It is a system for implementing windows on bitmapped displays. It implements a protocol for communications between 2 programs, a **server** which is responsible for drawing on a display and interpreting events from a mouse and keyboard and a **client** which is some application server that requests things be drawn on the screen (and which can be located on the same machine or somewhere else on the network).

Note that the client and server are the reverse of typical client server situations. For example, you may be running a large simulation package on a server somewhere while logged into a Linux workstation with an X display. The server in this case is the Linux workstation while the client is the simulation package on the server system. The client sends requests to the server to perform various draw operations using the X protocol which usually runs under another low-level protocol such as TCP/IP.

X does not specify how what the user interface should look like or how the window manager should respond to events. This behaviour and look and feel is left entirely up to client programs. This has resulted in great flexibility but a lack of a common look and feel for X. In recent times, Linux has introduced some standard components and toolkits which have resulted in 2 different standard look and feels - GNOME and KDE.

The free X11 code-base was split in 2004. Originally maintained by the Xfree86 group, they introduced a more restrictive license in 2004 which many X developers found unacceptable. A new organisation, the X.Org foundation was started and began working on a fork of the X11 code. Most distributions are in the process of moving to using the X.Org releases in current distributions. The practical difference to the user is small at the moment but X.Org is expected to introduce many improvements to the overall architecture of X over time.

Window Managers and Widget Libraries

- Xlib
- Window Managers
  - TWM
  - FVWM
  - Kwin
  - Metacity
- Client Applications
- Widget Libraries
  - Athena
  - Motif
  - GTK+
  - Qt

X provides a low-level library called Xlib which allows clients to make drawing requests to the server.

The X protocol does not dictate look and feel, or how to manage multiple windows or how to launch new applications. The designers of X wanted it to be flexible enough to allow individual clients to specify this behaviour. In practice, users prefer some consistency when interacting with a windowing system and various clients were developed which handle this task – these are called window managers. There are many different window managers available for X which provide different look and feels, different ways of managing windows and different features.

From a client applications point of view, it wants to be able to create windows and display some output in them with a minimum of work. In practice, the Xlib library is very low-level and makes basic tasks such as displaying some text or displaying a menu containing a series of buttons quite difficult. This resulted in developers creating libraries specifically for performing these tasks which could be used by developers of client applications, allowing the client applications to focus on their functionality be it displaying a calendar or a web page, rather than spending most of their time implementing low-level functionality.

The **Athena** (or **Xaw**)widget set was the original widget set used on X (actually developed by MIT as a sample widget set for developers but widely adopted in early X clients). It is basic and non user-friendly by todays' standards. It was followed by the **Motif** widget set which was introduced by a group of vendors and required licensing to use. It was user and programmer friendly but its cost ultimately discouraged Linux developers from using it in their projects.

The **GTK+** widget set was developed by a group of free software developers who were working on a free bitmap graphics editor for Linux and wanted a modern, free widget set for their use. They subsequently released the widget set under a liberal license for others to use.

The **Qt** toolkit was introduced as a commercial alternative to Motif by a Norwegian company called Trolltech. It was introduced with a dual license which made it free for use in GPL projects but required payment for use in commercial projects.

The 2 leading widget sets on modern Linux systems are GTK+ and Qt by virtue of their use in the main desktop environments. There are a wide variety of other free toolkits in existence and active development.

Desktop Environments

- Standard look and feel
- CDE
- KDE
  - Kwin window manager
  - Qt toolkit and KDE library
  - Kpanel launcher
  - Konqueror file manager
- GNOME
  - Metacity window manager
  - GTK+ toolkit
  - Gnome panel
  - Nautilus file manager

While a myriad of window managers and toolkits gives developers and users great choice in how they want to configure their systems, it does result in some problems.

- Users moving between systems may find themselves using a different window manager which performs differently to the manager they are used to.
- Applications written with different toolkits have radically different look and feels and modes of interaction.
- Each different library used on a Linux system resulting in more memory being used. If a user has 5 applications onscreen and each one uses a different toolkit, the memory used is much higher than if they all use the same library (due to Linux's use of shared libraries – a library only needs to be loaded into memory once even if multiple applications are using library components).
- There are also issues performing actions between applications using different toolkits like cut and paste.
- Finally, the kind of features that users have come to expect in graphical environments such as a control panel may not be available depending on the window manager in use.

Desktop environments were introduced to standardise some aspects of the desktop. They try to bring a common look and feel to the desktop and are usually built around one toolkit. Applications written for that desktop environment follow a set of usability guidelines and utilise the same underlying toolkit. The desktop environment also uses a standard window manager and provides some standard tools including a panel (like the Microsoft Windows start bar), a file manager, web browser and help system. A control panel is also usually integrated into the desktop environment.

Traditional UNIX systems used a desktop environment called the Common Desktop Environment (CDE) built around the Motif toolkit and providing a standard look and feel. On Linux systems, the 2 main desktop environments are KDE and GNOME. Both of them provide similar functionality and applications with similar functionality – their main differences from a user perspective are their look and feel and how certain tasks are managed on each. Modern distributions provide both environments allowing users to choose between them.

Installation and configuration

- Installation
  - Hardware is automatically detected
  - Choose desktop environment
  - Configuring display manager
- Configuration
  - SuSE / Novell Linux Desktop
    - yast2 (or sax2)
  - Red Hat
    - redhat-config-xfree86
  - Debian
    - dpkg-reconfigure xserver-xfree86
- Fonts

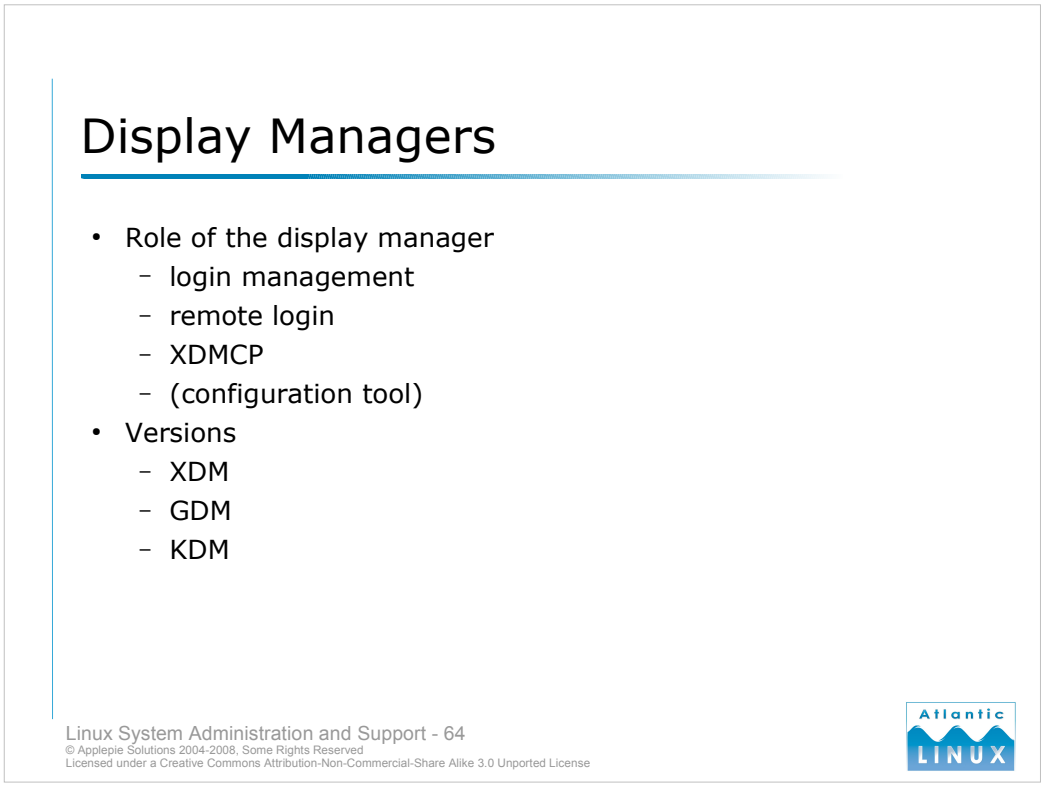
Traditionally, it was necessary to manually configure X when installing Linux on a system. Modern Linux distributions automatically detect your hardware and generate a suitable configuration without any user intervention. It may be necessary to review or manually tweak this configuration at some stage if problems occur though.

The configuration files are usually found in **/etc/X11**. The main X configuration file is **XF86Config** (for Xfree86 releases) or **xorg.conf** (for X.org releases), this configures up the various devices to be used by X including the graphics card, the monitor, the keyboard and mouse. It also specifies what resolutions to run at and what 3D configuration to enable.

X is normally started by a display manager (to be covered later) but can also be started by an individual user from the command-prompt using the **startx** command. When invoked in this manner, startx will attempt to read a local file called **.xinitrc**. The .xinitrc dictates what window manager should be used and can be used to automatically start some applications on login.

On SuSe / NLD, the only configuration of X to be done at startup involves choosing your default desktop environment (either GNOME or KDE) and optionally specifying what resolution and colour depth to use for your display. These settings can subsequently be changed through the Yast configuration tool. On Debian, you normally provide more details during configuration including explicitly specifying what graphics driver to use. Red Hat requires a similar level of configuration.

X can be configured to display fonts using different systems. Originally, a separate program called the X Font Server (**xfs**) was used to provide font rendering services to the X Server. This could be used by just the local machine or by remote machines also (so you could use one X Font Server for a group of machines). Recent versions of the X server program include a font renderer which eliminates the need to run a separate font server. The built-in font renderer is preferred for security and ease of configuration and handles display of traditional UNIX font formats such as postscript and bitmap and the newer TrueType font format (as used under Windows).



As discussed, you can start an X session from the command-line with the startx command. It also possible to have X start at system boot and allow any user to log directly into a graphical X session. The piece of software that manages this is called the display manager.

XDM

The original display manager, the X display manager (xdm) has a limited set of features. by default, it prompts for the username and password and logs the user in.

GDM and **KDM**

GNOME and KDE supply their own display managers which provide significant functionality over and above XDM. They allow you to choose what kind of session you wish to have before logging in. A session indicates what software should be used when a user logs in and can consist of either a desktop environment such as GNOME or KDE or just a window manager (such as TWM or FVWM). GDM and KDM can also use Themes to customise their appearance. Both GDM and KDM provide a graphical configuration tool rather than requiring you to edit a configuration file to make configuration changes.

XDMCP

The X Display Manager Control Protocol allows users to connect to an X server running on a different system, effectively providing remote graphical logins to the another system. Any of xdm, gdm or kdm can be configured to provide a user with a list of available remote systems for graphical login rather than a local login screen.

Exercise 5 – X

1. **Restart your session using a different desktop environment or Window manager and note some differences.**
2. **Change the resolution of your default desktop**
3. **Enable XDMCP on your display manager**
4. **Start the XDMCP chooser on your display manager and connect to another desktop**
5. **Identify the video driver being used in your X configuration (hint: Section is "Device")**

Software packages

Overview

- Overview
  - Red Hat (rpm)
  - Novell Desktop Linux / SuSE (rpm)
  - Debian (deb)
  - Slackware (tgz)
- Dependencies
- Versions
- Package contents
  - Packaged software
  - Installation software
  - Package information
  - Dependency information

Linux systems contain a wide range of programs and applications. You can install new software on a system by downloading the source (when available) and compiling it yourself but this is time-consuming and can be a maintenance nightmare (especially when you want to move to a newer version of the program while preserving your original configuration).

Linux distributions have taken a number of approaches to **software packages** in order to reduce this maintenance overhead and simplify the installation and upgrading of software packages. There are two main formats used for packaging software on Linux systems – **RPM** – a format introduced by Red Hat (and now used also by SuSE, Mandrake and a range of others) and **DEB** – a format introduced by the Debian project (and used by various Debian derivatives including Knoppix, Ubuntu, Progeny and Xandros). The original "packaging format" used by some distributions (and still used by Slackware) – **TGZ** is simply an abbreviation of **tar.gz** which indicates the technologies used.

In all cases, the software package usually contains the following:

- information about the package
- scripts to manage installation, upgrades and deinstallation
- the software to be installed which can be either binaries or source code
- details of any libraries or other pieces of software that the package requires to operate correctly.

Software packages are typically compiled against certain **versions of libraries** for a **specific architecture** – package managers will only allow the package to be installed on a system that matches these requirements. Some package managers (particular the Debian ones) automate the management of these dependencies upgrading packages to required versions automatically if needed.

RPM

- Installing
  rpm -ivh <*package*>.rpm
- Listing
  rpm -qa
- Removing
  rpm -e <*package*>
- Advanced uses
  rpm -qi <package>
  rpm -qR

Red Hat and SuSE use the RPM packaging system and the **rpm** command to manage packages. There are also various graphical tools for managing the packages on the system but it is useful to be familiar with the rpm command especially if you need to install 3rd party software on a Red Hat system.

To install a new package, *rpm -i <filename>* is the minimum required. Typically, *rpm -ivh <filename>* is used to provide some more feedback on the installation process.

If a version of the package is already installed, you can use the upgrade option instead of the install: *rpm -Uvh* which de-installs the other versions of the package after installing this version.

To review what packages are already installed on the system, *rpm -qa* is used.

To remove (erase) a package, use *rpm -e*.

The rpm command has a lot of more advanced features (the current man page for the rpm command runs to about 800 lines). The *-q* option can be used for more than just listing all installed packages, it can also retrieve information for individual packages including

-qi <package>	lists information about <package>
-qR <package>	lists dependencies for <package>
-ql <package>	lists all files in <package>
-qf <file>	lists the package which owns <file>
-q –scripts <package>	lists the scripts used by <package>

To run any of these commands against an uninstalled RPM, add *-p <filename>*

Advanced Package Management Tools

- Automatically download packages from network
- Resolve dependencies automatically
- Debian
  - apt
  - aptitude
  - synaptic
- RedHat
  - up2date
  - yum
- SUSE
  - RedCarpet

Most distributions now come with tools to automate the process of downloading packages and installing dependencies. These tools do not supersede the basic package management tools such as RPM – they simply automate the task of downloading and installing rpm (or .deb in the case of Debian) packages, and the dependencies of those packages. These tools also enable the system to automatically download updates to packages that are already installed, as they become available.

up2date is the tool used by most versions of RedHat, although it is being superseded by *yum*. Note that, in the case of RedHat Enterprise Linux, these tools are configured by default to download packages from the RedHat Network – as such, a RedHat network subscription is required.

apt-get is the tool used by Debian. Two variants in common use are *aptitude*, which has better dependency management than *apt-get*, and *synaptic*, which provides a GUI interface.

Newer versions of SuSE provide use a graphical tool called red carpet for automatically downloading new packages and updates from SuSE servers.

Red Hat Package Management Tool

- Graphical Tool for Managing Packages on Red Hat
  - Install
  - Remove
  - Update
  - Automated Dependency Management
- Invocation
  - Applications > System Settings > Add/Remove Applications
  - system-config-packages
- Package Groups
  - Standard packages
  - Extra packages

Red Hat also provide a graphical tool called the **Package Management Tool** for managing software packages if you are not comfortable using the rpm command. It provides a user-friendly interface allowing you to install, remove and update packages.

When you select a new software package for installation with the Package Management Tool, it will automatically identify any dependencies and mark these for installation also.

To start the tool go to the **Applications** menu on the main panel, select **System Settings** and **Add/Remove** applications. It is also possible to start the tool from the command line by typing `system-config-packages`. Inserting CD #1 of Red Hat Enterprise Linux will also cause the Package Management Tool to be started (if you are running with a graphical environment).

Packages are divided into **Package Groups** which categorise packages according to their purpose include *Administration Tools*, *Mail Server, Web Server, X Window System, KDE* and so on. Each group consists of **Standard Packages** that must be installed if the package group is installed and **Extra packages** that the user can optionally install for additional functionality.

Red Hat Network

- Automatic System Updates from Red Hat
  - Security Alerts
  - Bug Fix Alerts
  - Enhancement Alerts
- Red Hat Update Agent
- https://rhn.redhat.com/
- Automatically updates 1 or more systems
- Requires a valid Red Hat subscription

Red Hat provide an Internet based solution for managing software updates to systems with a valid Red Hat Subscription or License. If you have a valid subscription and are configured to use the Red Hat Network, you will receive emails when security alerts, bug fix alerts and enhancement alerts are issued by Red Hat for the version of their software that you are running on your systems. In larger organisations, Red Hat provide a version of the Red Hat Network that runs within the organisation's network and does not require you to connect to the Internet – details of how to connect to this should be available from your organisation's IT department.

Red Hat also provide a software agent which can be run on individual systems which connects periodically to the Red Hat Network and automatically updates the system.

To configure/update your system to be used with the Red Hat Network go to the **Applications** menu on the main panel, select **System Tools** and the **Red Hat Network** menu option. This will start the registration process.

Note that the Red Hat Network requires a valid Red Hat subscription. Red Hat subscriptions are normally purchased for 12-months. It is legal to continue using Red Hat without a valid subscription but you are not entitled to any updates or technical support from Red Hat.

Source packages

- Sources
  - author homepage
  - freshmeat.net and sourceforge.net
- Checksums
- Extraction
  - tar zxvf <package>.gz
  - tar jxvf <package>.bz2
- Development packages
- Steps
  - ./configure
  - make
  - make install (to /usr/local)

It is recommended to always use the distribution-supplied package for an application if one is available and meets your needs. In some cases, it may be neccesary to obtain a newer version of the application than is currently available in the distribution. In this case, you will need to download an updated package from the author of the software. Typically, you will need to download a simple package (tar file or compressed tar file) containing the source code for the package. You will then need to compile the package source and install the resulting package on your system.

After downloading the package from the author's website (or a site such as *http://freshmeat.net* or *http://sourceforge.net*) you will need to extract the package. Some sites also provide a checksum file. This can be used to verify that the package is an authentic package from the author and not a file that has been tampered with by a third party (to insert malicious code). You can verify a checksum using the **md5sum** command with the **--check** option and the checksum file. It will display an error if the package checksum does not match the one in the file.

Source packages are typically available in either **.tar.gz** or **.tar.bz2** format. Both are compressed tar-files. To extract them using **GNU tar**, use **tar zxvf <*filename*>.gz** or **tar jzvf <*filename*>.bz2**. This will create a source directory containing various files and subdirectories. It is convention to include a file called **INSTALL** or **README** which should contain instructions on how to configure and compile the package. Normally, the following steps will install a working package. You may also need to edit a configuration file for complex applications,

- `tar zxvf <package>.gz or tar jzxvf <package>.bz2`
- `cd <extracted package dir>`
- `./configure`
- `make`
- `make install`

The **configure** command in step 3 runs a script which verifies that any include files and libraries required to compile the application are available. If not, you may need to install these additional include files or libraries which can normally be found in your distribution. The convention is to name include files packages with a -devel so qt3-devel would be the package containing the qt3 include files. Libraries are usually named starting with lib so libqt3 would be the package containing the qt3 libraries (this may change a little from distribution to distribution and some packages may include versions in the package name).

Managing shared libraries

- Libraries
  - static
  - shared
- Portability of static binaries (e.g. skype)
- Commands
  - ldd <command>
  - ldconfig -v
- Files
  - /etc/ld.so.conf
- Variables
  - LD_LIBRARY_PATH
  - LD_PRELOAD

Many operating systems use libraries to provide common routines and code which many different programs can use. Examples of libraries include the qt library which contains the qt widget set and the GNU C library which supplies the basic C routines used by many Linux programs. There are 2 different types of libraries - static and shared (also known as dynamically linked). Static libraries are built as part of the program. Each program using a static library has its own copy of this library. A program using shared libraries does not include a copy of them, rather the program loads a shared library when it starts and other programs that require the library can use the same copy. This results in much reduced memory usage as only 1 copy of any shared library is in memory at any time.

Shared libraries do introduce some complexities though. A program cannot run on a system if the shared library has not been installed. Conversely, a program using a static library can be used on any system since it carries all the code it requires.

Problems can sometimes occur starting programs due to missing shared libraries or when the installed shared libraries are of a different version than that required by the program. You can check what shared libraries a program uses with the ldd command, for example,

```
$ ldd /bin/ls
        linux-gate.so.1 =>  (0xffffe000)
        librt.so.1 => /lib/tls/librt.so.1 (0x40024000)
        libacl.so.1 => /lib/libacl.so.1 (0x4002c000)
        libc.so.6 => /lib/tls/libc.so.6 (0x40032000)
        libpthread.so.0 => /lib/tls/libpthread.so.0 (0x4016a000)
        /lib/ld-linux.so.2 (0x40000000)
        libattr.so.1 => /lib/libattr.so.1 (0x4017d000)
```

This indicates that the ls commands depends on these shared libraries for operation. Note also the version numbers in the library names. The locations of shared libraries on a system are stored in a file called **/etc/ld.so.conf**. The **ldconfig** command reads this file to determine what libraries are available (**ldconfig -v** lists the currently available libraries).

If you add non-standard libraries to the system you should place them in one of these directories or add a new directory to the /etc/ld.so.conf. You may also use the environment variable LD_LIBRARY_PATH to indicate additional locations to search for shared libraries and the LD_PRELOAD variable to specify libraries to be loaded before all others.

Exercise 6 – Software packages

1. **List all installed packages**
2. **Display some information about the bash package**
3. **Display the dependencies the bash package has**
4. **Display the shared libraries that the bash binary uses.**
5. **Download and install the srm source package from sourceforge.net**

Atlantic LINUX

User and Group Management

System Accounts

- Users
  - uid
  - username
  - password
- Groups
  - gid
- /etc/passwd
- /etc/group
- Other authentication mechanisms

A user account represents someone or something capable of using files on the system (can be either a person or a system process).

A group account is a list of user accounts. Each user account has a main group and as many others as is needed.

Users are defined in the /etc/passwd file. This file contains various information about users including their **login name**, **encrypted password**, **uid**, **gid**, **user's real name**, **home directory** and **shell**. Modern systems tend to store the users password in a separate file (/etc/shadow).

Groups are defined in /etc/group. This file contains the **group name**, **group password**, **gid** and a **list of users**.

Users and groups provide the operating system with a way of controlling access to system resources and maintaining an audit trail.

Linux systems can also use more advanced authentication systems such as **kerberos** or **LDAP authentication** (e.g. Active Directory) by adding **Pluggable Authentication Modules** (**PAM**) to the system.

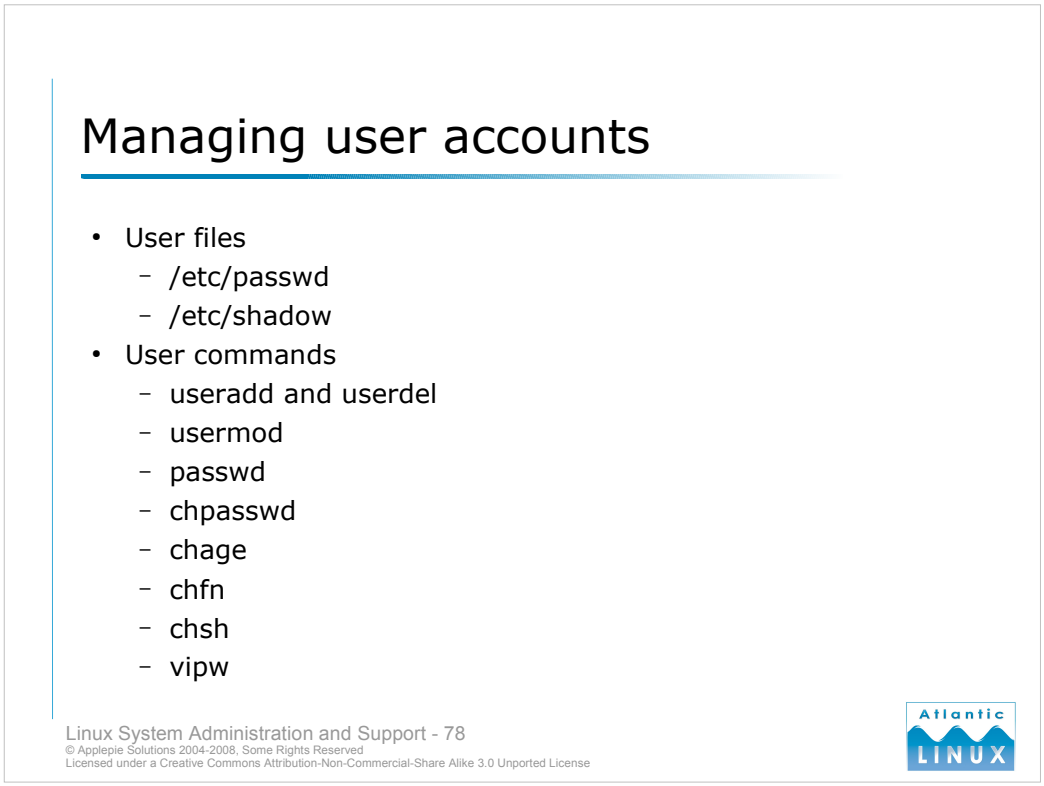
Account Settings

- Passwords
  - choosing good passwords
  - password expiry
  - password security
- Shell
- Home directories

Each user account has a password. Different systems enforce different password policies but a good password is generally hard to guess and not a simple word (like Password!).

The **shell** or **command interpreter** is the program that takes your commands and does something with them. All user interaction with a Linux system is conducted through the shell (if using the console, users of a graphical environment can interact with the system without using the shell although one will still be associated with the user). There are a number of different shells available on a typical Linux system which use slightly different syntax.

Each user account is assigned their own directory within which to store their files. This directory is known as the **home directory**. Its location varies depending on the particular Linux version (**/home/*username*** and **/usr/users/*username*** are common locations).



User details are stored in the **/etc/passwd** file. There is one entry per line, and each line has the format:

account:password:UID:GID:GECOS:directory:shell

Where,

- *account* is the name of the user on the system. It should not contain capital letters.
- *password* is the encrypted user password or *.
- *UID* the numerical user ID.
- *GID* the numerical primary group ID for this user.
- *GECOS* is an optional field uses for informational purposes. It usually contains the user's full name.
- *directory* is the user's home directory.
- *shell* is the program to run at login (if empty, */bin/sh* is used). If set to a non-existing executable, the user will be unable to login through login(1).

Traditionally, passwords have been stored in this file also. When UNIX systems were first introduced, most multi-user systems were friendly environments. Modern multi-user systems are more impersonal and modern hardware can be used to crack passwords if the encrypted version can be seen. Modern systems thus usually store the passwords in a separate file /etc/shadow which is only readable by root.

New users are added using the **useradd** command. Existing users are removed using the **userdel** command. The **usermod** command is used to change settings such as account expiration.

The **passwd** command can be used to change an individual user's password. The **chpasswd** command can be used to batch change a number of user passwords at the same time. The **chage** command is used to change the user's password expiry information.

The **chfn** command can be used to change the user's personal details and the **chsh** command changes the user's shell.

The **/etc/passwd** file can be edited directly with **vi**. There is a small danger of this file getting corrupted by multiple users editing it (or running commands such as useradd). There is a wrapper for vi called **vipw** which sets appropriate locks on /etc/passwd before opening it for editing. This ensures that multiple edits on the file will not cause any corruption.

Managing groups

- Group files
  - /etc/group
  - /etc/gshadow (*not used on SuSE/NLD*)
- Group commands
  - groupadd and groupdel
  - groupmod
  - gpasswd
  - grpck
  - vigr

Group details are stored in the **/etc/group** file. There is one entry per line, and each line as the format:

group_name:passwd:GID:user_list

Where,
- *group_name* is the name of the group.
- *password* is the (encrypted) group password. If this field is empty, no password is needed.
- *GID* is the numerical group ID.
- *user_list* is a list of the group member's user names, separated by commas.

Note that the group file also contains an optional password field. This is used if a user who is not a member of a group tries to change their GID to that group using the newgrp command. If the password exists, the user must provide this password before newgrp successfully changes their GID.

On some systems, as with **/etc/shadow**, the **/etc/gshadow** file is used to store the encrypted passwords to protect them from cracking attempts. SuSE/NLD does not currently use /etc/gshadow.

The **groupadd** and **groupdel** commands are used to add and remove groups. The **groupmod** command is used to modify the name or GID of a group.

The **gpasswd** command is used to set or remove passwords from groups.

The **grpck** command is used to validate the /etc/group file.

As with /etc/passwd, you can edit /etc/group with vi but there is a small danger of file corruption if multiple users edit the file at the same time. The **vigr** command is a wrapper around vi which sets appropriate locks to prevent this corruption.

Shell configuration files

- Bourne shells
  - /etc/profile
  - ~/.profile
  - ~/.bash_profile
  - ~/.bash_login
- C shells
  - /etc/csh.login
  - /etc/csh.cshrc
  - ~/.tcshrc
  - ~/.cshrc
- .bashrc (interactive and non-interactive shells)
- . and source

When a shell is started, it reads a configuration file which can be used to customise the environment and perform housekeeping tasks on behalf of the user. Typical actions performed might include customising the shell prompt and maybe starting a mail monitoring program such as biff. **Environment variables** can also be set in this file.

Each shell has their own particular configuration file (derivatives of the **bourne** and **csh** shells generally read either their own config file or their ancestors config files).

sh
- **/etc/profile**
- **~/.profile**

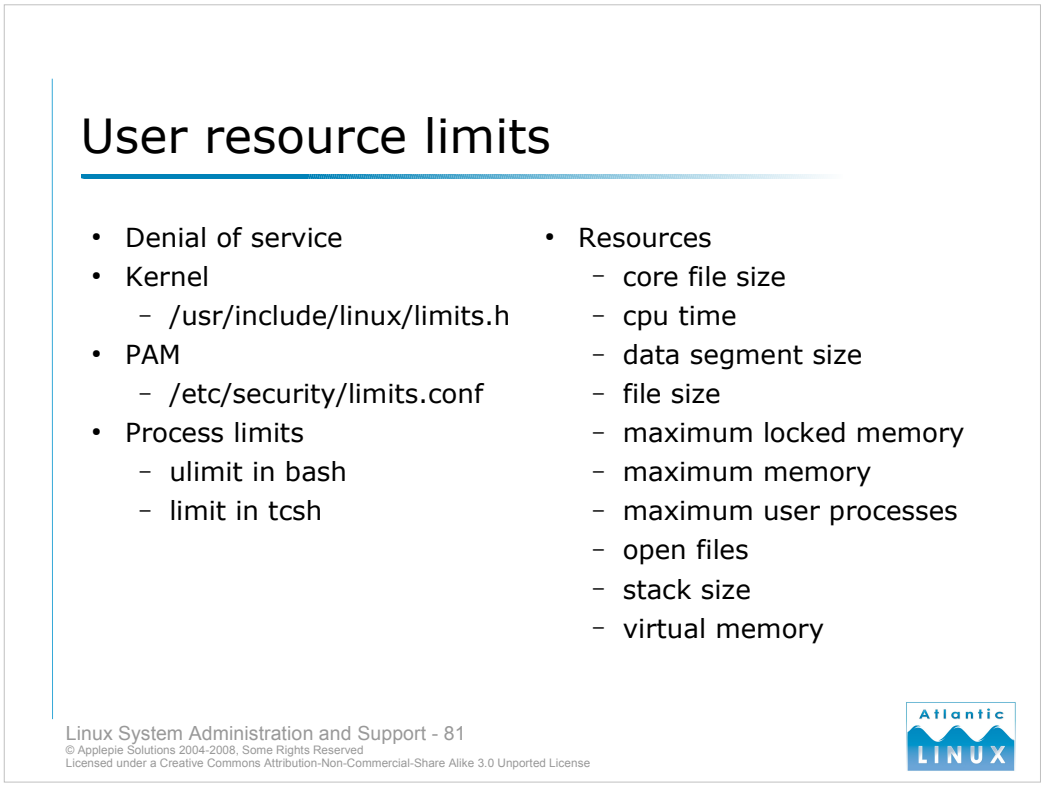
bash
- **/etc/profile**
- **~/.bash_profile**
- **~/.bash_login**
- **~/.profile**

csh
- **/etc/csh.cshrc**
- **/etc/csh.loginc**
- **~/.cshrc**

tcsh
- **/etc/csh.cshrc**
- **/etc/csh.login**
- **~/.tcshrc**
- **~/.cshrc**

Note that there are global configuration files under /etc. The shell typically reads these before reading the per-user configuration files in the user home directories. Environment variables such as PATH (and other system-wide variables) should be set in the files in /etc. These variables can be reset or extended by users in their individual configuration files.

The **source** or **.** commands can be used to read or re-read these configuration files.



Each user on a Linux user can consume various system resources. System resources include things such as open files (filehandles), memory allocated for processes and total cpu time used. There are finite amounts of each of these things available on a Linux system.

You may wish to restrict resources for individual users for a number of reasons. On large multi-user systems, you will not want any user to be able to excessively impact on other users on the system. You may also wish to restrict the amount of memory that system processes such as webservers can allocate in order to prevent problems if such a process is hacked or overloaded (in a so-called *Denial of Service attack*). The amount of each of these resources that an individual user can consume can be controlled in various ways. You can see some of the hard limits that are built-in into the kernel by looking at **/usr/include/linux/limits.h**.

The security subsystem used on Linux, PAM, provides a module for restricting user resources at login time. Resources can be restricted on a per-user or per-group basis and you can set hard or soft limits for each resource. Hard limits are absolute limits beyond which a process cannot use any additional resources. When a user exceeds a soft limit, they will normally receive a warning of some sort. Users can extend their soft limits up to the hard limit (they basically provide a mechanism for users to limit their own resource usage but push it out when required).

The bash and tcsh shells also come with commands for manipulating these resource limits. The shell built-in command on bash is ulimit and the command in tcsh is limit (and unlimit). These allow the following resource limits to be reconfigured,

- **core file size** - the size of a core dump file created if a program crashes
- **cpu time** - the amount of cpu time that any process can use (in seconds)
- **data segment size** - the maximum size of a processe's data segment
- **file size** - the largest single file which can be created
- **maximum locked memory** - the maximum size which a process can lock into memory using mlock
- **maximum memory** - the maximum amount of physical memory a process can have allocated to it
- **maximum user processes** - the maximum number of processes this user id may have open simultaneously
- **open files** - the maximum number of open files for this process
- **stack size** - the maximum stack size for this process
- **virtual memory** - the maximum amount of virtual memory available to the shell

Scheduling jobs and managing user access

- Running jobs once at some future time
  - at
- Running jobs once when the system is under-used
  - batch
- Running jobs regularly
  - cron

```
0 7 * * * ~/bin/daily-backup.sh
0 7 * * 1 ~/bin/weekly-backup.sh
0 7 1 * * ~/bin/monthly-backup.sh
```

- Controlling access
  - allow
  - deny

Linux provides a number of facilities to run commands at some time in the future without requiring user interaction at that time. The two main approaches use the **at** command and the **cron** facility.

The **at** command is intended to run commands once at a particular time, for example, if you wanted to start a large compilation during the middle of the night when it wouldn't inconvenience other users of the system.

An at job is scheduled using the *at time* syntax where time can be in a number of formats including *HH:MM*, midnight, noon, teatime, *month-name day year*, *MMDDYY, MM/DD/YY, MM.DD.YY*, *now + time* (where time is specified in units of *minutes, hours, days or weeks*).

The at command then prompts for the command(s) to be run and input is terminated with CTRL-D. Any output resulting from the command is emailed to the user. Pending at jobs an be viewed with the **atq** command. Jobs can be removed with the **atrm** command.

e.g.

```
# at midnight
at> touch /var/tmp/i_ran_at_midnight
at> CTRL-D
job 1 at 2005-08-10 00:00
#
```

The **batch** command is a variant of at which runs a scheduled job when the system load falls below 1.5

The **cron** facility is used to run recurring tasks on a periodic basis, for example, system backups are usually run through the cron facility. Jobs are scheduled by adding an entry to the users **crontab** file with the command *crontab -e*. The format of the file is

<minute> <hour> <day of month> <month> <day of week> <command>

You can restrict use of the cron facility using **allow** or **deny** files. The location of these differ from distribution to distribution (NLD uses /var/spool/cron/deny and /var/spool/cron/allow while Debian uses /etc/cron.deny and /etc/cron.allow). If the allow file exists, only users listed in this file will be allowed to use cron. If the deny file exists, users listed in the file will not be allowed to use cron. The at command uses a similar system (with /etc/at.deny and /etc/at.allow being the files used on both distributions).

Exercise 7.1 – User and group management

1. **Add a new user to the system**
2. **Change the user's home directory to /home/newhome and move the existing user directory**
3. **Add a second user to the system**
4. **Write a script to set the passwords of both new users to a new string**
5. **Remove this user from the system**
6. **Using the chage command, force one of the new users to change their password at next login.**
7. **Change the users shell to /bin/tcsh**
8. **Open 2 shells on your system as root. Start vipw in one. Leave that vipw session running and start a vipw session in the other window. Note any errors.**
9. **Add a new group.**

Atlantic LINUX

Exercise 7.2 – User and group management

10. **Change one of the users to be a member of this group.**
11. **Limit the file size that can be created for one of the new users to 1MB.**
12. **Login as this user and attempt to create a larger file (use dd /dev/zero ...) and note any errors.**
13. **Schedule a regular job to back up your home directory to a compressed archive in /var/tmp. The job should run at 5:00am every day except Sunday. Ensure no-one else can view or extract the file.**
14. **Explain the following cron entries**

```
25 4 * * 1 w
* * * * * /bin/monitor.sh
0 * * * * /bin/wall /0.txt
5 9-17 * * 1-5 /bin/work
```

Process Management

Processes and Threads

- Heavyweight process
- Lightweight process
- Speed of context switch
- Speed of creation
- Ease of sharing data
- Security
- NPTL

Multitasking operating systems use the concept of a **process**. A process is simply an instance of a running program. More modern systems have introduced the concept of **heavyweight processes** and **lightweight processes**. Traditional processes are the same as heavyweight processes. Lightweight processes are also known as threads.

A process can be said to consist of **code, data** and at least one **thread of execution**. Early UNIX process models contained only one thread of execution. Each process used its own **virtual address space** securely separating its code and data from all other programs executing on the system.

This separation comes at the price of a relatively large amount of time required to create new processes (since the memory needed by the process needs to be organised). In addition, when a system **context switches** between executing programs, it can take a relatively large amount of time to unload the data and code of a process and load the new process's data and code. Running multiple threads in the same context eliminates some of these problems.

Linux combines these two approaches – its **fork()** system call uses **copy on write** semantics to minimise the cost of process creation (a child process is initially given a pointer to its parent's memory which reduces the startup and context switching costs).

In addition to the lightweight process model, Linux provides a number of other threading libraries including a **Native POSIX Thread Library (NPTL)** for Linux. This attempts to address some of the shortcomings found in trying to map traditional UNIX thread models to the Linux model with early versions of Java for example.

Shell job control

- foreground jobs
- background jobs (&)
- listing jobs
- switching
- suspending
- interrupting

Most shells include at least some basic **job control** which allows a user to create and manage multiple processes from a single prompt.

When a command is executed at the prompt, it is said to run in the **foreground**. The shell suspends and won't process any more input until a foreground job finishes (a job can be **interrupted** using CTRL-C). A job can also be **suspended** using CTRL-Z.

A command can also be run in the **background**. The command runs as normal but the shell will continue to accept additional input and can run other commands while the background jobs complete. A job can be run in the background by putting the & symbol at the end of the command. The shell normally displays the **job number** and **process number** when a background job is started.

The **jobs** shell command lists out all running and suspended background jobs.

A particular job can be brought to the foreground using the **fg** shell command and the job number preceded by the % symbol. A suspended background job can be resumed using the **bg** shell command.

Listing processes

```
# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
 root         1  0.0  0.0   1492   104 ?        S    Aug28   0:04 init [2]
 root         2  0.0  0.0      0     0 ?        SW   Aug28   0:00 [keventd]
 root         3  0.0  0.0      0     0 ?        SWN  Aug28   0:00 [ksoftirqd_CPU0]
 root         4  0.0  0.0      0     0 ?        SW   Aug28   6:05 [kswapd]
 root         5  0.0  0.0      0     0 ?        SW   Aug28   0:24 [bdflush]
 root         6  0.0  0.0      0     0 ?        SW   Aug28   0:10 [kupdated]
 root       222  0.0  0.0      0     0 ?        SW   Aug28   0:10 [kjournald]
 root       223  0.0  0.0      0     0 ?        SW   Aug28   0:00 [kjournald]
 root       294  0.0  0.0      0     0 ?        SW   Aug28   0:00 [khubd]
 daemon    1508  0.0  0.0   1604    64 ?        S    Aug28   0:00 /sbin/portmap
 root      1564  0.0  0.2   1628   360 ?        S    Aug28   0:56 /sbin/syslogd
 root      1603  0.0  0.0   2152    80 ?        S    Aug28   0:02 /sbin/klogd
 root      1607  0.0  1.8  12644  2324 ?        S    Aug28   0:00 /usr/sbin/named
```


The **ps** command provides a more complete view of the processes on a Linux system. Using ps, you can view all processes on the system rather than just those commands that you have executed from the prompt.

The ps command takes a large number of options to customise its output and include various fields. **ps ux** gives a reasonably informative listing of a users' processes including various details. **ps aux** uses the same output format to display all processes on the system.

Two other options to note are **f** for **forest** and **w** for **wide**.

ps aux columns

- **USER** is the **user name** of the process owner.
- **PID** is the **process ID**, a number that uniquely identifies a process (for the lifetime of that process).
- **%CPU** The average processor usage of the process.
- **%MEM** The percentage of physical memory used by the process.
- **VSZ** The amount of virtual memory allocated to the process in kilobytes (This will typically be much larger than the actual memory usage).
- **RSS** The resident set size – the amount of memory actually in use by the process.
- **TTY** The terminal (if any) that the process is attached.
- **STAT** The current **process state** (see next slide).
- **START** The start time or date of the process. This is the 24 hour clock time for the first 24 hours and the start date after that.
- **TIME** The amount of time this process has been executing on the CPU for.
- **COMMAND** The command that the process is executing ([thread]).

Process listing variations

- `ps -elf`
- `ps`
- `ps u`
- `ps ux`
- `ps x -o user,pid,ppid,cmd`

Process states

- Runnable (R)
- Sleeping (S)
- Uninterruptible Sleep (D)
- Traced or stopped (T)
- Defunct or zombie (Z)
- Additional BSD status codes
  - No resident pages (W)
  - High priority process (<)
  - Low priority process (N)
  - Process with pages locked in mem (L)

A process can be in various states on a Linux system. Typically the **kernel scheduler** gives each process a short period of execution time and then gives the next process a chance. This period of execution time is known as a **quantum** or **timeslice**. On Linux, it is normally about 100ms (but this varies widely depending on the kernel version, the system hardware and the scheduler in use).

When a process is ready to to be executed, it is added to the **run queue** and its state is set to **runnable** (**R**). If a process requests some resource which isn't currently available, such as access to an I/O device, it is put to **sleep** (**S)** until this resource becomes available.

Processes in **uninterruptible sleep** (**D**) have marked themselves as uninterruptible while performing some critical task (typically used by device drivers)or accessing a resource that must not be left in an unknown state. When the process is finished with this resource, it removes the uninterruptible flag. This flag is typically used by kernel related threads manipulating internal buffers. A process which appears in a **D** state in a process listing is usually suffering from an I/O problem of some sort – processes should only become uninterruptible for very brief periods of time while accessing I/O etc.

Stopped processes or processes being traced will be flagged as (**T**).

Zombie (**Z**) or **defunct processes** occur when the parent process of a child process has not called the wait() system call. A zombie process is actually only an entry in the process table which won't be cleared until wait() is called – they don't use any memory or cpu. A large number of zombie processes may indicate a problem on the system (parent processes dying for some reason).

Monitoring processes

- top cpu processes
- cpu state information
- similar details to ps
- process priority and nice

The **top** command is another useful alternative for monitoring processes running on the system. It provides a continuously updating list of the top processes in terms of cpu usage.

Top displays a section of statistics relating to CPU and memory usage and a list of the active processes on the system. It displays similar information to ps.

The **PID** field as the same as ps, providing the process id for that particular process. The **USER**, again, is the owner of the process. **PR** is the priority of a task. Tasks with a numerically lower priority value are given preferential treatment by the task scheduler.

The **nice** command can be used to change the priority of a process. The **NI** column reflects any changes made to a process using this approach.

VIRT is the size of a processes **virtual image** in kilobytes. It includes a processes code, data, shared libraries and any memory pages which have been **swapped out**. The **RES** column lists the **resident size** of a process in kilobytes. This is the physical memory currently in use by a task. The **SWAP** column lists the value of memory pages which have been swapped out in kilobytes.

$$VIRT = SWAP + RES$$

S is the process state as per ps.

%CPU and **%MEM** show the percentage of processor usage and physical memory usage of a process.

Signals

- What are they?
- What do they do?
- Sending signals via the keyboard
- Sending signals with the kill command
- Sending signals with system calls
- Common signals

The operating system uses a mechanism called **signaling** to send short messages to processes when it wishes to notify a process of an important event. Signals interrupt the normal execution of a process and force it to **handle** the signal immediately on receiving it. Typical events which might require a signal to be sent to a process include **floating point exceptions**, **termination signal from the user**, **suspend signals from the user**, **I/O errors** and so on. Signals are defined as integer values. A full list of signals is available in *signal(7)*.

Processes handle signals by passing execution to a **signal handler** routine which performs some specific activity in response to the signal. Typical actions a signal handler might perform include killing child processes and removing temporary files before terminating its own process. Some signals cannot be caught (SIGKILL, SIGSTOP).

When you use CTRL-C to interrupt a job or CTRL-Z to suspend a job, you are actually sending that process a signal (**SIGINT** in the case of CTRL-C and **SIGTSTP** in the case of CTRL-Z).

Arbitrary signals can be sent to any process using the **kill** command. The normal syntax is *kill <signal> <PID>*. Running kill with the **-l** option lists the available signals. A common use of kill is to send a SIGINT to a runaway process,

```
kill -9 1234
```

You can resume a suspended process using either the **fg** or **bg** commands. Both commands send a **SIGCONT** to the process.

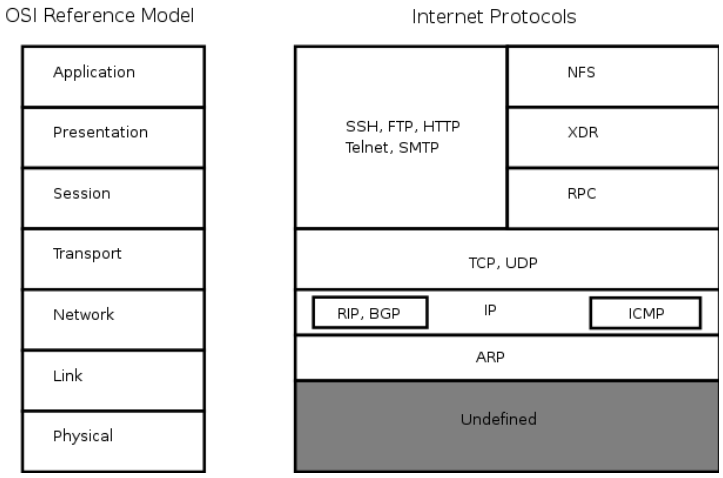
Signals can also be sent programatically using the **signal()** system call..

Exercise 8 – Processes

1. **Review the man page for ps and do the following:**
   - **display only your processes**
   - **display all processes**
2. **Display processes in a hierarchy showing parent and child processes (hint: forest).**
3. **Identify some high priority tasks running on the system.**
4. **Identify some processor and memory intensive tasks.**
5. **Start a simple process of your own and experiment with running it in the background and bringing it to the foreground (echo).**
6. **Kill the process while it is running in the background.**

Network Configuration

Networking Concepts



Networks consist of **hosts** i.e. members, usually computers but possibly other participating devices such as printers. A collection of hosts on a network is sometimes referred to as a **site**.

All hosts on a network, as well as being connected by some medium either wired or wireless, must also agree on a set of protocols for communicating with each other. The OSI model is an abstract reference model for communications and computer network protocol design. The internet protocol suite commonly used on the internet roughly maps to the OSI model as shown in the diagram.

The everyday protocols you use are at the top (application layer). Examples include **ssh** (for connecting to machines), **ftp** (for transferring files), **http** (for browsing the web) and **smtp** (for sending email).

Underneath these applications, either TCP or UDP is used to actually transport data packets. Tranmissions Control Protocol (TCP) is a connection-oriented protocol that offers guaranteed delivery of packets (with an overhead in network handshaking that this requires) while User Datagram Protocol (UDP) is a connectionless protocol that transmits packets on a best effort basis without any guarantees about delivery.

TCP and UDP run on the Internet Protocol which describes the protocol used to encapsulate and transport data around the network. It handles details such as addressing and routing. Dynamic routing of IP packets around the network is handled by protocols including RIP, BGP and OSPF. Internet Control Message Protocol (ICMP) is used to send status and error messages around IP networks (this is the protocol used by tools such as ping).

At the lowest level, the Address Resolution Protocol (ARP) is used to map IP addresses back to the physical addresses used by the underlying hardware. In the case of Ethernet networks for instance, each network device has a unique physical address (the MAC address). ARP is used to map these addresses to the IP addresses used at the logical network level.

The internet protocol suite can run on a variety of physical networks and media including Ethernet, Token Ring and Wireless..

IP Addresses

- Address (network part and host part)
- Netmask (used to identify the network part)
- Network
- Broadcast (address all hosts on a network)

Address	**192.168.0.1**	11000000.10101000.00000000. 00000001
Netmask	**255.255.255.0 = 24**	11111111.11111111.11111111. 00000000
Host part	**0.0.0.255**	00000000.00000000.00000000. 11111111
Network	**192.168.0.0/24**	11000000.10101000.00000000. 00000000
First Host	**192.168.0.1**	11000000.10101000.00000000. 00000001
Last Host	**192.168.0.254**	11000000.10101000.00000000. 11111110
Broadcast	**192.168.0.255**	11000000.10101000.00000000. 11111111

Networks consts of **hosts** i.e. IP identifies each host on the network with a 32-bit IP address (in IP v4), e.g

11000000 10101000 00000000 00000001

For convenience, these addresses are usually represented as 4 decimal numbers with each number mapping to a byte of the 32-bit address (ref: **ipcalc** command)
e.g.

192 168 0 1

This is usually split into a **network part** and a **host part**. So an example network might be 192.168.0 which contains a maximum of 254 hosts (192.168.0.1 to 192.168.0.254). 192.168.0.0 is the **network address** and 192.168.0.255 is the **network broadcast address**.

The netmask is used to identify the network part of an IP address.

The broadcast address for a network can be used to directly address all devices on a particular network e.g.

```
ping -b 192.168.0.255
```

sends an ICMP message to all active hosts on the 192.168.0 network.

Devices and Tools

- Network devices
  - eth0, eth1, ...
- Tools
  - ifconfig
  - ping
  - telnet
  - traceroute
  - route
  - ipcalc

Linux network devices do not appear in /dev. Network devices are created by the kernel device drivers when they recognise network hardware. They are named *eth[0..n]* in the order they are discovered by the operating system.

The **ifconfig** command is used for querying and configuring network devices – actions possible include assigning addresses and netmasks to devices, querying work devices and activating and de-activating network devices e.g.

```
ifconfig eth0 10.0.0.1 netmask 255.255.255.0
```

This would configure the first network device (*eth0*) with the address 10.0.0.1 and set the netmask to 255.255.255.0.

Running ifconfig with the **-a** option or with a specific network interface returns the current configuration for that device.

The **ping** command can be used to verify connectivity to a particular system on the network. It takes a hostname or IP address as an argument and attempts to send one or more test packets to that host. It reports whether or not it succeeds and provides statistics on the quality of the connection.

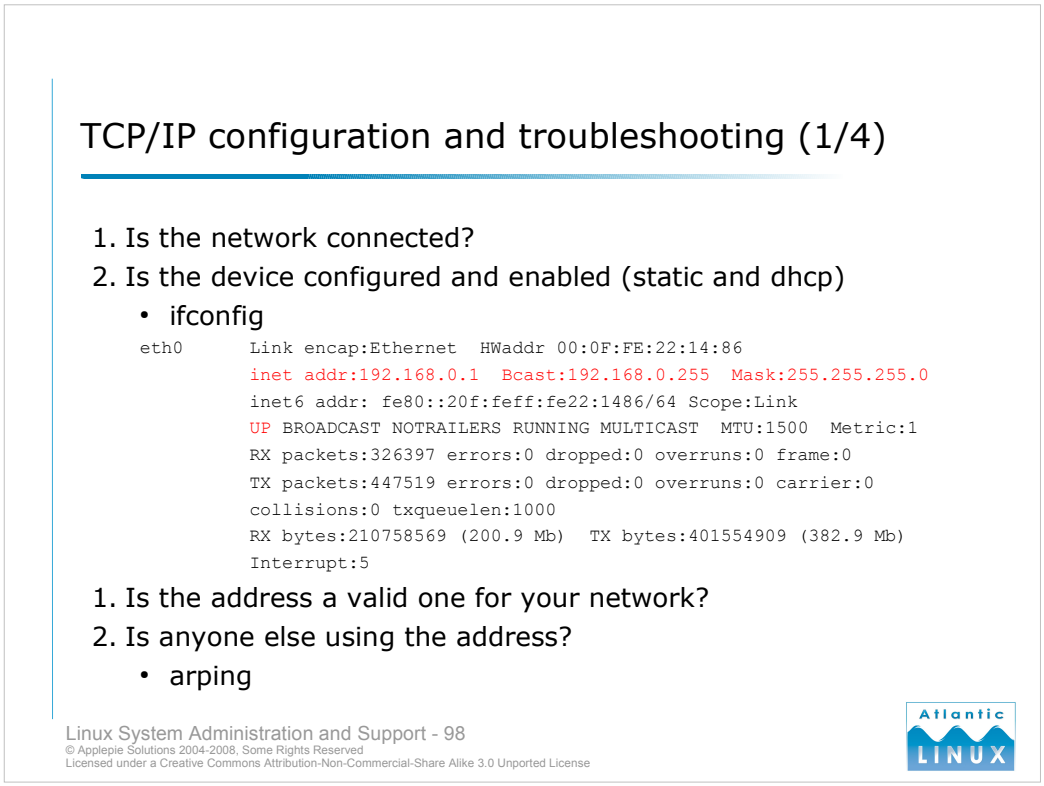
The **telnet** command can be very useful for testing text network protocols (as well as being a tool for connecting to systems) e.g.

```
telnet www.example.com 80
```

Connects you to the webserver port on www.example.com allowing you to send standard HTTP messages and review the servers response. In general, it is recommended to use ssh rather than telnet for actually connecting to systems due to telnets use of cleartext passwords on the wire.

The **traceroute** can be used to see the actual path across a network that packets are taking to a certain destination. It can be useful as a first step to diagnosing a routing problem.The **route** command is used to view or set routing information (for static routes).

The **ipcalc** command (*http://jodies.de/ipcalc*) is useful for testing network and netmask configurations.



The checklist above is a suggested strategy for diagnosing network problems. The basic principle is to work up the network stack and eliminate the fundamental causes of a problem first.

1. You should first verify that the network card is physically connected to the network. This includes verifying that the network cable is physically plugged into the device. Most network cards also have 1 or more LED lights close to the socket for the network cable. One of these lights is the "link light" which indicates if the network card has a live connection to another network device (normally a switch or hub). The link light should be illuminated if the card is connected and working properly. The card may also have a second light, the "activity light" which illuminates when traffic is travelling on the device, you should see this light briefly when using the ping command for instance.

2. You should next confirm that the card is configured up. A network device will normally either be configured with static settings or using dynamic addressing (where the address and other details are supplied by a DHCP or BOOTP server). Use the **ifconfig** command to check what network settings have been assigned to the device. If the *inet addr*, *bcast* or *mask* fields are empty, the device is incorrectly configured. You should also see the keyword *UP* in the list of flags. The **ifdown** and **ifup** commands are used to shutdown or start a network device using the existing configuration. It may be useful to stop and restart the device to review any error messages that are generated during configuration. If you are using DHCP, you should check the system logs for errors when the DHCP service is started on the client.

3. You may also need to review your network settings with the network administrator, it is possible that you are using the wrong network settings for the network the device is connected to.

4. You may also experience problems if your chosen address is in use by another device on the network. To check if this is the case, perform the following steps (this example assumes your interface is *eth0* and that you have the **arping** tool installed)

```
ifdown eth0
ifconfig eth0 up
```

[run arping to see if any MAC address responds with your IP address]

(NLD) `arping -D <your ip address>`

(Debian) `arping <your ip address>`

If you receive a response for the ip address then someone else is using your IP. You will need the assistance of the Network administrator to identify the problem system or allocate a new address.

TCP/IP configuration and troubleshooting (2/4)

5. Are the netmask and broadcast correct?
6. Are the routes correct?
   - Check with route -n

```
$ route -n
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         192.168.0.1     0.0.0.0         UG    0      0        0 eth0
```

   - Check a path with traceroute

```
$ traceroute -n www.google.com
traceroute to www.google.com (66.102.9.104), 30 hops max, 40 byte packets
 1  192.168.0.1  0.160 ms   0.141 ms   0.154 ms
 2  192.168.0.200  0.603 ms   0.818 ms   0.784 ms
 3  169.131.100.25  20.560 ms   32.542 ms   41.286 ms
 4  83.71.114.145  49.295 ms   57.287 ms   62.779 ms
```


Atlantic LINUX

5. Verify the netmask and broadcast address with your network administrator or by using the ipcalc tool.

6. Your system will normally be configured with a set of network routes to local systems. If you are having connectivity problems, it may be a routing issue. You can review your current routes with the route command. When called as "route", it displays the current routes using DNS names. It may be advisable when troubleshooting to use "route -n" which does not attempt DNS name lookups, displaying only the IP addresses.

The routing table lists the destination for traffic and the gateway that is used for traffic to that destination. The destination should be read in conjunction with the destination. So, for line 1, traffic with a destination of 192.168.0.0 and a mask of 255.255.255.0 will be matched and sent to the 0.0.0.0 gateway (traffic that matches this line would include traffic for 192.168.0.1 and 192.168.0.5 but not 192.168.1.1).

The 169.254.0.0 route is added automatically by SuSE/NLD and is used for auto-configuring networks where DHCP is not working.

The 0.0.0.0 destination is a special route called the **default route**. The default route is the route used for traffic when nothing else matches, in this case, the default route is to a local router with the address 192.168.0.1. Note that you can also set and remove routes with the route command.

The **traceroute** command can be used to check the path your data is taking to a particular destination (or where it is being lost). In the case above, packets are not routed beyond 83.71.114.145 which suggest a problem with the routing on this system. If this system is outside of your network, you may need to contact the administrator of that system to further diagnose the problem.

TCP/IP configuration and troubleshooting (3/4)

7.Testing connections with ping
- Ping command
  - basic test of connection
  - measure of network latency
  - measure of packet loss
- Connections to test
  - own system
  - system on local subnet
  - local gateway
  - system on another subnet
  - system on the internet

Atlantic LINUX

7. The **ping** command is a basic IP network diagnostic tool. It uses the **ICMP protocol** to send a small packet to a test destination (the ICMP ECHO_REQUEST datagram). If the host receives the packet, it should respond with another ICMP packet (the ICMP ECHO_RESPONSE datagram). This is a good measure of basic connectivity. The ping tool reports when it receives a response and generates some statistics including the **packet round-trip time** (which can be a rough measure of network latency) and the **packet loss** being experienced on the network (which an indicate a network problem). Ping recognises duplicate or damaged packets both of which may indicate a problem with the network.

Start testing with ping by pinging your network device IP address to verify that this is configured. If this works, ping another system that you know to be up on the local network. If this does not work, it seems likely that you have a problem between your system and the network switch or between your system and the other system.

If you can ping your local subnet, ping your gateway (the one specified as the gateway for your default route). If this does not respond, you will not be able to send any traffic out of your subnet. Verify that the gateway is the correct address and that it is configured to accept traffic from your host. If the gateway responds, try pinging a system on another subnet in your organisation to verify the basic operation of the gateway.

As a final ping test, you may wish to ping a well known internet host to verify overall connectivity. Be aware that many large organisations do not allow ping traffic out of their organisation in which case this will fail. Many large internet systems may be configured to silently ignore pings which may also cause a false negative result.

TCP/IP configuration and troubleshooting (4/4)

8. Are there firewalls in use on the network?
   - Are they between you and the problem system
   - Have they logged any messages for your systems IP
9. Is this a DNS problem?
   - Are there DNS servers defined in /etc/resolv.conf?
   - Can you connect ok to an IP address?
   - Are your DNS servers working (dig or nslookup)?
10. Is the MAC address getting correctly mapped to the IP?
   - arp -v

8. When testing network problems, you should be aware of any firewalls in use on your network (including any that may be running on the system you are testing). A firewall could block some or all traffic to/from certain destinations resulting in some difficult to understand diagnostic results. If you have access to systems with running firewalls on the network, you may want to verify that they are not logging any errors while you are conducting your testing - it may be that they are accidentally blocking or dropping some traffic from your problem system.

9. A DNS query involves sending a request to an external server (known as a nameserver or DNS server). The list of servers to query is stored in */etc/resolv.conf*. If multiple servers are specified, the system will query each one in turn until it find one which responds. You should verify that you can ping at least one of the DNS servers.

DNS queries can be conducted using either the nslookup or dig commands. nslookup is not recommended for diagnosing DNS problems (is has been superceded by the dig tool). The basic syntax for dig is,

`dig <hostname>`
for checking the IP address of a system when we know the system name.
`dig -x <IP address>`
for checking the hostname of a system when we know the system IP address.

10. The **arp** tool lists the table of MAC addresses and IPs. If this table contains corrupt or missing MAC addresses then you may be experiencing a problem at the physical media level.

Domain Name System - DNS

- Overview
- nsswitch.conf
- /etc/hosts
- /etc/resolv.conf
- host
- dig
  dig <hostname>
  dig -x <IP address>
- nslookup
  nslookup <hostname>
  nslookup <IP address>

The **Domain Name System** (**DNS**) is a system for mapping human-readable names to the IP addresses used by systems on the Internet. DNS misconfiguration is a potential source of problems and malfunctions in systems and system applications.

Linux systems use a number of sources for naming information – the system stops searching for DNS information when it finds an answer. The order in which a system searches for DNS information is controlled by the **hosts** line in */etc/nsswitch.conf*. It is normally configured as

```
hosts    files dns
```

This tells the system to look for DNS information in the **hosts file** first and, if that fails, to send a **DNS query** to a known DNS server.

The **hosts file** is a simple text file stored in */etc/hosts* which stores a list of IP addresses and hostnames in the following format,

```
IP_address canonical_hostname aliases
```

A **DNS query** involves sending a request to an external server (known as a **nameserver** or **DNS server**). The list of servers to query is stored in */etc/resolv.conf***.**

DNS queries can be conducted using either the **nslookup** or **dig** commands. **nslookup** provides a simple default interface for querying names but has some limitations (in particular, it does a name lookup on a DNS server before running your query which sometimes produces spurious error messages). **dig** provides far more detail by default and is a more flexible tool.

Network services

- Super-daemons
  - inetd
    - /etc/inetd.conf
    - /etc/services
    - disable by commenting line and restarting
  - xinetd
    - /etc/xinetd.conf
    - /etc/xinetd.d
    - disable by adding disabled line to config
- Standalone daemons

As a network operating system, Linux includes facilities for providing network services. These include services such as a webserver, a mailserver and so on. The /etc/init.d mechanism allows daemons or services to be started when the system is started. Each service that is started creates a small amount of overhead for the system, even though these services may only be required occasionally or not at all.

The **inetd** or **xinetd** *super-daemon* was introduced to address this problem. It is started at system startup through the /etc/init.d mechanism. It is configured to listen to various ports used for other services. For example, it can be configured to listen on port 80 for HTTP (web browser) requests. When it receives a request, it will take care of starting the service required to service the request (in this case a webserver). This reduces the system load at start-up and makes maintenance of new services more straightforward. It is normal to see a combination of services on a system with some being run through inetd/xinetd and some being run as standalone daemons. Some applications recommend that they are run as standalone daemons to reduce their response time (the Apache webserver being a typical example) especially if you are expecting a large amount of continuous traffic for the service.

inetd was the original implementation. It is configured via the /etc/inetd.conf and contains details of each service which it is managing. Note that it uses a service name which maps to an entry in the /etc/services file which is intended as a definitive reference of what services use particular ports (and whether they are udp or tcp services). Note that ports below 1024 are normally only usable by root.

To disable an inetd service, you should comment out that service in the /etc/inetd.conf and send the SIGHUP signal (kill -HUP <PID>) to the inetd to trigger a reload of the configuration file (you can find the PID of the inetd by running ps aux | grep -i inetd).

xinetd is an advanced version of inetd and is intended as a more functional, secure replacement for inetd. Debian uses inetd by default while Novell Linux Desktop uses xinetd. xinetd contains an overall configuration in /etc/xinetd.conf and also allows each new service to install a configuration file in /etc/xinetd.d. It does not use /etc/services.

To disable a service in xinetd add a line of the form "disabled = <service>" to the /etc/xinetd.conf and sending a SIGHUP to the xinetd to trigger a reload of the configuration.

Mailserver overview

- Terminology
  - MTA
  - MSA
  - MUA
- Software
  - Sendmail
  - Postfix
  - Exim
- Configurations
  - Local mail only (*this should never be disabled*)
  - Satellite System
  - Internet mailserver

Linux is often used as a Mail Server. Mail is usually sent using the Simple Mail Transport Protocol (SMTP). SMTP is used to send mail between Mail Transer Agents (MTAs) - the mail server software. Mail Storage Agents (MSAs) provide access to email to end-users. Normally, an end-user queries an MSA using a protocol such as POP3 or IMAP. The software used by an end-user for sending and receiving mail is called the Mail User Agent (MUA).

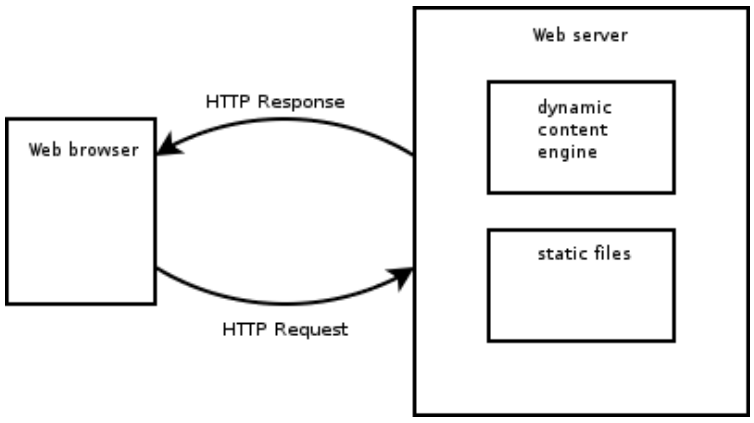
The definitive MTA for UNIX systems including Linux is Sendmail - a large powerful MTA which can be configured to do pretty much anything with email. Unfortunately, this power comes at a price and sendmail is difficult to configure and secure. As a result of this, it has mostly been superceded on Linux platforms by other MTAs. The 2 most popular MTAs in use on Linux systems are postfix and exim. Postfix was written from the ground-up as a secure, easy to administer replacement for sendmail. It should work as a drop-in replacement for sendmail in most standard installations. Exim is also a drop-in replacement for postfix. It has a very straightforward configuration but does not scale as well as postfix or sendmail making it more suitable for smaller sites. SuSE/NLD uses postfix by default. Debian defaults to exim but also provides a postfix package.

When a Linux system is installed, it will install an MTA by default. The MTA will normally default to providing local delivery. This means that mail is sent and received between users on the system but not sent outside of the system. Most Linux systems require this level of mail functionality to operate normally, a lot of system processes rely on the ability to send mail between each other - you should not disable local mail functionality.

If you are placing the Linux system into an existing environment that has a working email infrastructure that supports SMTP, you should be able to configure your system as a satellite mail system by specifying an outgoing mail server. This will result in the mailserver handing email for users outside of the local system over to this outgoing mail server.

In the absence of an outgoing mail server, you can configure the system to send and receive email via SMTP, your server will function as a full internet mailserver capable of sending and receiving email from others. Note that you will need correct DNS records to ensure that mail for your domain is sent to your system. You will also need to configure outgoing email so that it uses a valid internet address in the From address.

Webserver overview



A web server is a piece of software running a system which responds to requests from clients running web browsers. Requests are usually sent from the client to the server using the **Hyper Text Transfer Protocol (HTTP)** or **Hyper Text Transfer Protocol with Secure Sockets Layer (HTTPS)**. HTTP is a simple text protocol that for requesting data from a web server. HTTPS is a secure version of HTTP which encrypts the requests and responses so that only the client and server know what is being transmitted. HTTP uses tcp port 80 and HTTPS uses tcp port 443.

HTTP can be tested using the telnet command, for example,

```
# telnet www.nuigalway.ie 80
Trying 140.203.7.37...
Connected to www.nuigalway.ie.
Escape character is '^]'.
GET /
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>National University of Ireland, Galway (NUI Galway)</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
   8859-1">
<meta name="htdig-keywords" content="">
<meta name="robots" content="index, follow">
<meta http-equiv="cache-control" content="no-cache">
<meta http-equiv="pragma" content="no-cache">
....
</html>Connection closed by foreign host.
```

This can be very useful for diagnosing basic webserver problems and illustrates the simplicity of the protocol.

The web server itself is responsible for sending data back to the client when it receives valid requests. Data can be either static file content or dynamically generated pages. Static files are documents that are stored on the webserver's filesystem and that are sent to the client as-is. Dynamically generated pages are usually created on the fly using some 3rd party module (such as PHP or ASP).

The Apache Webserver - Introduction

- Versions
  - 1.3
  - 2.0
  - 2.2
- Configuration location
  - Debian
  - SuSE
  - Red Hat
- Main configuration file
  - httpd.conf
- Controlling apache with apachectl

The Apache web server is available in 3 major versions - 1.3 and 2.0 and 2.2. Apache 1.3 is maintained with bug fixes and security updates but is not under active development. Apache 2.0 was released in 2002 and introduced a more sophisticated threading model and various enhancements to make it suitable for use on Windows as well as UNIX systems. Apache 2.2 is an incremental upgrade to Apache 2.0 – this is the version that the Apache Foundation recommend for new deployments.

Debian provides both Apache 1.3 and 2.2 packages. Novell Desktop Linux does not include Apache in its default collection of packages but it can be installed with the Red Carpet package manager if the server software channel is selected. SuSE also offer both Apache 1.3 and Apache 2 packages. Redhat Enterprise Linux 4 defaults to Apache 2.0. This is the version that is covered by these notes.

The location of the Apache 2 configuration files is dependent on how Apache has been installed.
- By default, if you install Apache from source, it is installed to */usr/local/apache2* and the configuration files can be found in */usr/local/apache2/conf*
- On Debian, the configuration files can be found in */etc/apache2*
- On Red Hat and SuSE the configuration files can be found in */etc/httpd*

Older versions of Apache used 3 configuration files,
- *httpd.conf* which included the main configuration for the web server
- *access.conf* which included access settings for the web server
- *srm.conf* which included resource settings for the web server

Later releases of Apache have combined the contents of these files into 1 file - *httpd.conf*. These slides will assume all configuration is done in httpd.conf. Some distributions of Apache 1.3 may still require you to edit all 3 so if a particular setting cannot be found in httpd.conf, please refer to the other files. The Debian project have restructured the configuration directory layout for their distribution of Apache 2, so each site hosted on the server has its own configuration file (settings common to all sites are stored in apache2.conf).

Apache comes with a tool called **apachectl** which can be used to **stop**, **start** and **reload** the apache webserver (this is normally what scripts in /etc/init.d use under the hood to control apache).

The Apache Webserver – httpd.conf (1/5)

- Sections
  - Global environment
    - MPM Specific Settings
  - Main server configuration
  - Virtual hosts
- Global environment
  - ServerRoot
  - LockFile
  - PidFile
  - ScoreBoardFile
  - TimeOut
  - KeepAlive
  - MaxKeepAliveRequests
  - KeepAliveTimeout
  - Listen
  - LoadModule

The Apache configuration file is split into 3 sections. The first section specifies behaviour for the Apache server process as a whole (the Global environment). The second section specifies the behaviour of the main webserver and the third section specifies the behaviour of virtual hosts.
Each section contains a set of *directives*. The most important directives are explained in the following slides.

Global Environment
ServerRoot - this specifies the location of apache files on the system. It is used as a prefix for any other locations specified in the configuration file that do not begin with a '/'.
LockFile - this is used by Apache to manage incoming connections requests and ensure only one server thread attempts to respond to a request at any time, it should only be changed from the default if the default points to a file on shared storage such as NFS.
PidFile - this is used to record the PID of the apache server process and should not be changed.
ScoreBoardFile - this is used for internal Apache processes and should not be changed.
TimeOut - this is the time in seconds before an inactive connection to a browser is closed. This should not be set too low or browsers may find themselves not getting a response if they are on a slow network or the server is busy.
KeepAlive - It is possible for a browser to open a connection to a server and keep the connection open for multiple requests (thus saving the performance overhead of opening multiple connections). This directive indicates whether the server should allow browsers to request this functionality (values are *on* or *off*).
MaxKeepAliveRequests - this is the number of KeepAlive requests allowed in one connection.
KeepAliveTimeout - this is the time in seconds before an idle KeepAlive connection is closed.
Listen – Instructs Apache to listen on the given IP address/port. If only the port is specified, Apache listens on that port on all interfaces. Multiple Listen directives can be used, to enable Apache to listen on multiple addresses/ports.
LoadModule - Apache is built with a modular architecture. Most of apache's standard functionality is provided using standard modules, with only a small amount of functionality in the core itself. It is possible to add additional modules written in languages including C, Perl and Tcl. The LoadModule directive is used to load these modules into the server. You may see lines of the form,

```
<IfModule mod_status.c>
  ExtendedStatus On
</IfModule>
```

in the httpd.conf - these check if a particular module is loaded (mod_status in this case) before enabling that functionality.

The Apache Webserver – httpd.conf (2/5)

- MPM Specific Settings
  - MPM is core module for handling OS specific aspects of server
  - Several different MPMs available for Linux
  - MPM is determined at compile-time
    - Use *httpd -l* to determine what MPM is being used by apache
  - The following parameters should generally be changed if moving MPM:
    - StartServers
    - MinSpareServers
    - MaxSpareServers
    - MaxClients
    - ServerLimit
    - MaxRequestsPerChild

MPM Specific Settings

A major architectural change introduced in Apache 2.0 was the introduction of *Multi-Processing Modules* (MPMs). Each Apache server must run exactly one MPM. The MPM is responsible for low level operations such as handling operating system threads (covered in the section on process management). Different MPMs are used on different platforms (e.g. Linux vs. Windows). Several different MPMs are available for Linux, each with different performance characteristics.

The MPM to be used by Apache must be determined at compile-time. This means that if Apache is installed via an rpm or up2date, the MPM has been decided on by the package maintainer, and it can only be changed by finding an alternative rpm, or re-installing apache from source.

The MPM used on most distributions of Apache2 is called *prefork*, although Debian defaults to an MPM called *worker*. To determine which MPM is being used by a running Apache server, use the command *httpd -l*. This lists all of the modules installed in the Apache server, including the MPM (e.g. *prefork.c* or *worker.c*).

In general, *worker* provides better performance than *prefork*, but some Apache modules are not thread-safe, and these modules require *prefork*. At the time of writing, these are the only stable MPMs on the Linux platform, although there are several others in active development. Unless there is a specific performance or stability benefit to changing, it is best to use the default MPM that comes with your distribution's Apache package.

The MPM is configured using global directives. Some directives are specific to certain MPMs. Although the directives below are used by both *prefork* and *worker,* the recomended values of the parameters vary between the two MPMs. The default httpd.conf file supplies sensible defaults for both MPMs.

StartServers, MinSpareServers, MaxSpareServers - Internally, Apache has a number of server processes/threads running, each of which can service a web request from a browser. This allows the server to quickly respond to new web requests rather than having the overhead of starting a new process each time a request is received. MinSpareServers specifies the minimum number of such servers that Apache should have available for new requests and MaxSpareServers specifies the maximum number of such servers it should have available (servers will become available in the pool after servicing a request). StartServers specifies how many webservers to initially start with.

MaxClients - this specifies the total number of servers which should be running at any time. This is the absolute limit on the number of clients which can simultaneously connect to the webserver.

ServerLimit – The maximum value for *MaxClients*. This value must be >= *MaxClients*!

MaxRequestPerChild - this limits the lifetime on each server process to this many requests. It is intended to avoid problems with long running processes in situations where Apache itself or libraries it is using leak memory.

The Apache Webserver - httpd.conf(3/5)

- Main Server Configuration
  - User
  - Group
  - ServerAdmin
  - ServerName
  - DocumentRoot
  - <Directory ... >
  - AccessFileName
  - <Files ... >

Main Server Configuration

User, Group - Apache usually needs to be started by root since it needs access to ports lower than 1024. It is more secure to run the webserver as a less privileged user (to reduce the access an attacker gets to the system in the event of a break-in). User and Group identify the username and group that apache should be run as after starting.

ServerAdmin - the email address which should be used on error pages generated by the webserver.

ServerName - this specifies the hostname and port that apache identifies itself as. It allows you to over-ride the default system name with an alternate one (for example, your webserver may be called foo.example.com and have a DNS alias of www.example.com. You can use ServerName to ensure that responses come from www.example.com rather than foo. Note that you will need a correct DNS setup to allow this to work).

DocumentRoot - the directory from which static files are served.

<Directory .. > - You can specify what features and options are allowed for each directory to which apache has access. By default, all subdirectories of a particular directory which has a <Directory ...> option specified inherit these features and options. You can specify a separate <Directory ...> directive to customise the behaviour for the subdirectory.

AccessFileName - you can configure apache to restrict access to certain directories on the basis of either the IP of the client or a password supplied by the client. AccessFileName specifies the file in each directory which contains these settings.

<File .. > - As well as specifying access controls with the <Directory .. > directive, you can control access at the granularity of individual files using <File .. >. It supports similar options to <Directory .. >

The Apache Webserver – httpd.conf (4/5)

- Main Server Configuration (contd.)
  - HostnameLookups
  - ErrorLog
  - LogLevel
  - CustomLog
  - LogFormat
  - Alias
  - ScriptAlias

HostNameLookups – If this is set to *on,* hostnames of clients connecting to the server are logged. If it is off, clients' IP addresses are logged instead. From a performance point of view, it is much better to turn this off and only resolve the addresses when analysing the log-files.

ErrorLog – specifies the location of the apache error logfile.

LogLevel – specifies what messages to log in the error logfile (one of *debug, info, notice, warn, error, crit, alert, emerg* in order of increasing importance or reducing verbosity). The level you specify includes any levels that follow it also so specifying warn as the LogLevel will include warn, error, crit, alert and emerg.

CustomLog – specifies a location for a second apache logfile containing custom information. This logfile might be used to log who accessed the site, what browser they were using and maybe what site they came from.

LogFormat – this specifies the format to use in the CustomLog. Note that you need a CustomLog directive for each specified LogFormat.

Alias – this lets you map a directory on the webserver to a custom alias. For example, you have apache configured with a document root of */var/www*. You have some files in */var/www/docs/finance* which you wish to provide on your website. To make the url to these documents more useful, you decide to Alias it to */finance*. This allows users to access this information using the friendly url, *http://<your webserver>/finance*.

ScriptAlias – this works the same as Alias except that it is used for directories containing executable scripts. When a browser access files located in a ScriptAlias'ed directory, the webserver will attempt to execute them and return the results to the browser.

The Apache Webserver – httpd.conf (5/5)

- Virtual Hosts
  - Name based virtual hosts
  - IP based virtual hosts

```
<VirtualHost 10.1.2.3>
    ServerAdmin webmaster@support.example.com
    DocumentRoot /www/docs/support.example.com
    ServerName support.example.com
    ErrorLog logs/support.com.com-error_log
    TransferLog logs/support.example.com-access_log
</VirtualHost>
```


Virtual Hosting refers to the practice of serving multiple websites from the same physical machine and webserver. This allows a company to host multiple websites on one machine or indeed, for an ISP to host many different company websites on one machine.

There are 2 types of virtual hosts - name based and ip-based. With name based virtual hosts, your webserver uses one IP address but has multiple DNS names pointing at this IP address. This is the recommended form of virtual hosting to use. There are some reasons why you may not be able use name based hosting,

- You wish to manage bandwidth usage of individual sites using a tool which can only do so if the sites use different IP addresses.
- Your customers are using very old browsers which do not understand name-based hosting (name based hosting relies on the browser sending a request for a particular named host).

The following is a simple example of a configuration of name-based virtual hosting,

```
NameVirtualHost *
<VirtualHost *>
ServerName www.domain.tld
DocumentRoot /www/domain
</VirtualHost>
<VirtualHost *>
ServerName www.otherdomain.tld
DocumentRoot /www/otherdomain
</VirtualHost>
```

With IP based virtual hosting, you must configure multiple IP addresses for your webserver. These may be attached to different physical network devices or you may use virtual network interfaces (a feature of Linux which allows you to associate multiple IP addresses with one physical device).

```
<VirtualHost 10.1.1.1>
ServerName www.domain.tld
DocumentRoot /www/domain
</VirtualHost>
<VirtualHost 10.1.1.2>
ServerName www.otherdomain.tld
DocumentRoot /www/otherdomain
</VirtualHost>
```

The Samba server - Introduction

- Windows interoperability
- SMB/CIFS protocol
- Features
  - File services
  - Print services
  - Domain services
- Web based configuration interface (SWAT)
- smbclient tool
  - -L <samba server>
  - //<samba server>/<sharename>

The Samba server is a piece of software originally written for UNIX and Linux systems which supports the SMB and CIFS protocols. The SMB/CIFS protocol is used to share files and print services between systems. It is the native protocol used by Windows operating systems for file and print services. Linux systems using Samba can both use Windows shares and provide File and Print services for Windows systems.

Samba has a number of additional features for Windows interoperability,

- Operates as an SMB server for providing file and print services to clients Microsoft operating systems.
- Works as a Windows NT 4.0 domain controller replacement.
- Functions as a member of a Windows NT 4.0 or Active Directory domain.

Samba can be configured using a variety of means. The samba configuration files are usually installed in **/etc/samba** and can be edited using an editor such as vi. The Samba project also supplies their own web-based configuration tool called **SWAT**. Finally, NLD and other flavours of SuSE provide a Yast module for configuring Samba server.

Samba also provides tools for Linux systems to talk to Windows servers. The main tool is **smbclient**, a command-line tool for querying and using Windows shares.

The Samba server – Security modes

- security = user
- security = share
- security = domain
- security = ADS
- security = server

Samba supports a number of different security modes depending on how you want to fit it into your Windows infrastructure and how you want users to authenticate,

security = user
This is the default setting on Samba 3.0 and later. With this mode, any user connecting to the samba server must log-in with a valid username and password before they can use the server. With this security mode, the username and password are required before access is provided to any facilities on the server (including public access folders). Samba can use any of a number of password databases for authentication.

security = share
With this mode, people accessing the server do not require a username and password to connect to the server. Passwords are only checked when a specific share is being accessed. This allows Guest access shares to be easily configured.

security = domain
The username and password are passed to a Windows Domain server for validation (you still require an account for the user on the Samba server). The samba server must be a member of a Windows domain for this to work.

security = ADS
Samba acts as a domain member in an Active Directory domain (not as a Domain server).

security = server
This mode is very similar to security = user. The Samba server uses another SMB server (either Samba or Windows NT) to authenticate a username and password, falling back to security = user mode if this fails. This mode is not recommended.

The Samba server – Configuration (1/2)

- Samba daemons
  - smbd
  - nmbd
  - winbindd
- /etc/samba/smb.conf

```
[global]
workgroup = AWORKGROUP
netbios name = ASERVER
[share1]
path = /tmp
[share2]
path = /my_shared_folder
comment = Some random files
```


The Samba server software consists of 2 or 3 daemons depending on the configuration.

The **nmbd** (NetBIOS name server) handles all name resolution requests that occur as part of Samba browsing. This listens on the network for requests from clients that require a share from a particular server. If it recognises the server name for a share, it will respond to the client.

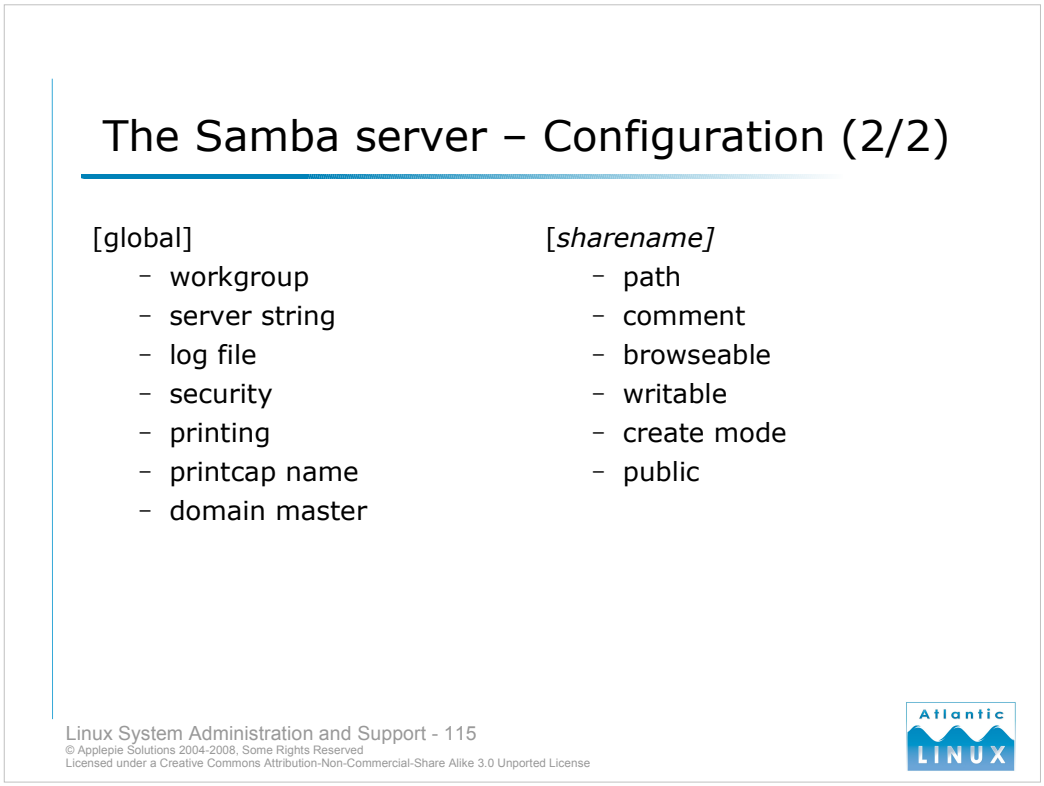
The **smbd** (SMB/CIFS server) handles all SMB protocol requests. This provides the file and print sharing services that are requested by clients and is at the core of the Samba server. It is also responsible for local authentication.

The **winbindd** (name service switch daemon for NT servers) is used if Samba is operating as part of a NT or AD domain. It handles authentication of users against NT and AD servers.

The main samba configuration file is **/etc/samba/smb.conf**. This specifies the behaviour of Samba including what security mode to use and what files and printers to share. Samba configurations range from very basic to extremely complex depending on what items you wish to share and what behaviour you want the server to have. A very minimal example is shown above.

Note that the smb.conf consists of sections which start with a token like *[section name]* where section name is either the keyword **global** or a **share name**. The keyword global marks a section which contains settings applicable to the entire samba server (and would include details such as what security mode to use, what domain the server is a member of and so on). Each share has its own section which includes details of the share including what directory on the filesystem the share maps to and what options should be applied to that share.

In practice, production samba configurations are slightly more complex than this but it is not unusual for samba configurations to run to less than 20 lines.



We will look here at the most important samba configuration directives. Note that you should be comfortable with the smb.conf man page if you are deploying a production samba server as there are large security implications in how the server is configured. The global section contains settings that apply to the entire server while each share section contains settings specific to that particular share. We will look at some of the most important settings. Note that there are many more settings which can be used with samba. The smb.conf contains an exhaustive listing of these. Most settings have a sensible default value (which is again described in the smb.conf man page).

Global

workgroup - this is used when clients query to indicate what workgroup the server is in. When security is set to domain, it is also used in the authentication.

server string - this is a short description displayed in browse lists of clients. it is optional.

log file - this configures where samba logs debug and activity messages. It can take a number of different formats and can be used to create a per-client log.

security - one of user, share, domain, server or ADS as discussed on the previous slide.

printing - this controls how print status information from the printing system is interepreted by the samba server. It can be one of bsd, aix, lprng, plp, sysv, hpux, qnx, soft or cups and should match the actual type of print system in use.

printcap name - this tells samba what printcap file to use for the printing system. The printcap file normally contains a list of available printers and other printer configuration details. If you are using a CUPS printserver, this should be set to the string cups as cups uses a non-standard printcap.

domain master - this controls the workgroup browsing behaviour of the nmbd. This should normally be set to auto to prevent network browse problems.

Share-specific

path - the path to the directory to share

comment - a string to display a description of the the share

browseable - this controls whether the share is displayed on the list of available shares for a machine.

writable - this controls whether users can write to a share.

create mode - a unix "chmod" style mask which controls what permissions are set on files created on the share.

public - this allows non-authenticated users to connect to this share. Note that this requires a guest account to be set in the global configuration and it does not work with security modes that require the user to authenticate before connecting to specific shares.

Jakarta Tomcat - Introduction

- Functions
  - Servlet container (Catalina)
  - JSP compiler
  - Implements Servlet and JSP specifications
  - Provides a standalone webserver
- Versions
  - 5.x
  - 4.x
  - 3.x
- Requirements
  - Java Development Kit 1.2 or later
  - Apache Ant tool

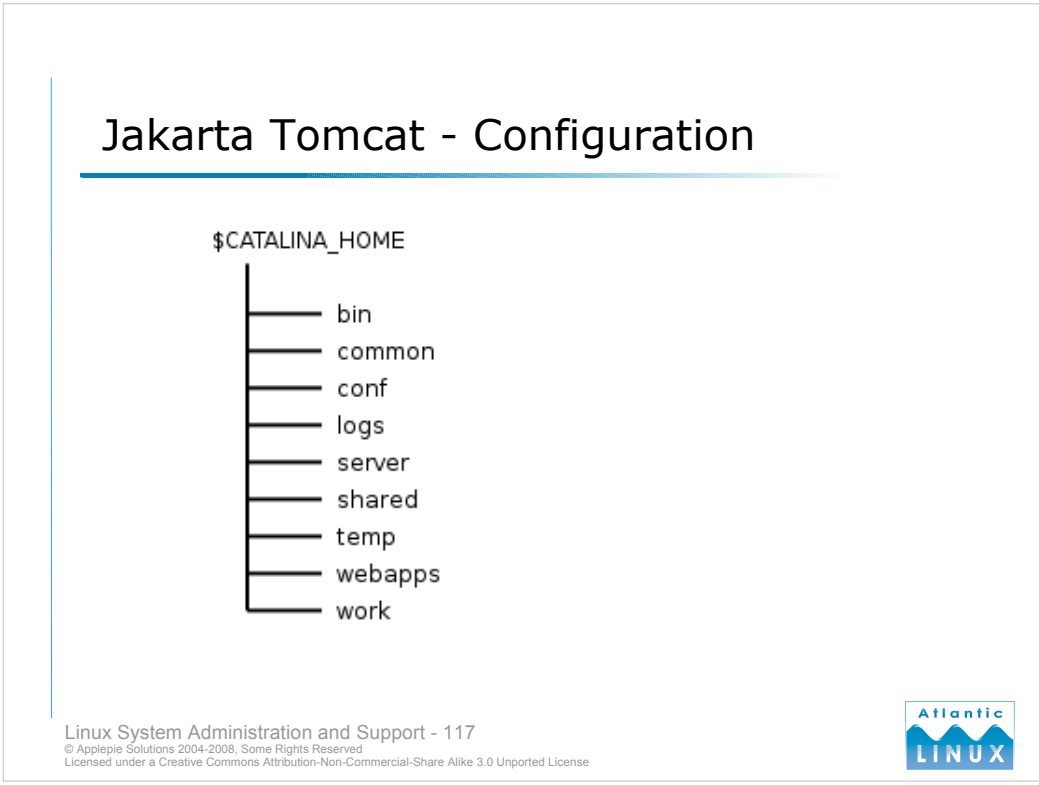
Tomcat is an application developed by the Apache foundation (the same group that provide the Apache webserver). It is part of the Jakarta project which develops open source Java components.

Tomcat is used as a servlet container. This allows it to run Java Servlets (a Java program that runs on a Web server and extends the server's functionality by generating dynamic content in response to Web client requests) and Java Server Pages (a form of dynamically generated HTML pages which use underlying Java code). Tomcat may be required to run 3rd party software, especially web-based or database driven applications.

Tomcat includes a webserver. Traditionally, Tomcat has been installed with the standard Apache webserver as its performance was not regarded as being good enough for production webserver use. This is not the case with modern versions of Tomcat and it should functional adequately as a webserver. This simplifies Tomcat configuration since it can be configured to webserve on the standard HTTP port (80). If used in conjunction with Apache, Tomcat will need to be configured to use a different port for its webserver (8080 by default for HTTP and 8443 by default for HTTPS).

Tomcat is currently available in a number of versions. Note that the different versions implement different versions of the Servlet and JSP specification. Tomcat 4.x introduced a new Servlet container called Catalina. Tomcat 5.x introduced faster JSP parsing and better garbage collection. Generally, unless constrained by 3rd party software, you should use the most recent stable release of Tomcat downloaded from the Jakarta Project's website.

Tomcat requires the JDK and the Ant tool to be installed. Tomcat will work with JDK 1.2 or newer but newer (1.4 or 1.5) versions should be more stable and are preferred.



The directory to which Tomcat is installed is referred to in the Tomcat documentation as $CATALINA_HOME (which is also an environment variable which should be configured to point to the directory in which Tomcat is installed).

A standard Tomcat installed consists of the following directories,

- bin - Contains startup, shutdown and other scripts and executables
- common – Contains common classes that Catalina and web applications can use
- conf - XML files and related DTDs to configure Tomcat
- logs - Catalina and application logs
- server - classes used only by Catalina
- shared - classes shared by all web applications
- temp – temporary files
- webapps - directory containing the web applications
- work - temporary storage for files and directories

Applications are installed into Tomcat under the webapps subdirectory. Applications should normally use the following organisational structure in a subdirectory under webapps,

- *.html, *.jsp – the top-level pages used by your application should be placed here.
- WEB-INF/web.xml – this is a configuration file which describes the servlets in your application and other configuration details.
- WEB -INF/classes – this is for class files in the applications that are not distributed in jar files.
- WEB-INF/lib – this is for components of the application that are distributed in jar files.

Note that common java libraries for use by more than one application can be placed in either of,

- $CATALINA_HOME/common/lib – jar files placed here are usable by all applications running on Tomcat and by Tomcat itself
- $CATALINA_HOME/shared/lib – jar files placed here are usable by applications running on Tomcat but not by Tomcat itself.

Tomcat applications are also distributed as Web Application Archives (also known as WAR files since this is normally their extension). If these are copied into the webapps directory, when started, Tomcat automatically extracts them. Note if you are upgrading such an application, you will need to remove the WAR file and the extracted subdirectories.

Network File System – NFS (1/2)

- Introduction
- Versions
  - 2
  - 3
  - 4
- User-space versus kernel NFS server
- Commands
  - mount
  - showmount
  - rpcinfo

The Network File System (NFS) is a system for sharing files between UNIX/Linux systems, similar to what Samba does for Windows systems. It allows directories from a system to be mounted on another system and treated as if they were part of the local filesystem. NFS consists of clients and servers. The server is the system making one or more filesystems available and the client is the system accessing the filesystem. A system can function as both a client and a server at the same time, sharing out directories to other clients and, in turn, mounting directories from other servers.

Version 2:
- Uses UDP in most implementations which results in bad performance on loaded or broken networks.
- Uses 32 bit filesizes which limits files to about 4GB.
- Data transfer size limited to 8KB which limited overall transfer speed of large files.

Version 3:
- Allows the use of TCP
- Uses 64 bit filesizes for much larger file sizes.
- Supports asynchronous writes for improved performance.

Version 4:
- Introduces better congestion control.
- Provide improved security.
- Has improved file-locking schemes.
- Uses a new file model which provides better interoperability with non-UNIX operating systems.

Support for NFS, version 3 was introduced in the 2.4 series Linux kernel and is the norm on systems today. Some work has been done on NFS v4 for Linux but it is not yet ready for production use. Standard distributions come with all the necessary components to support v2 and v3. Linux provides 2 different NFS servers - a server which runs entirely within the kernel and a server which runs in userspace. In practice, the kernel based server is slightly faster but the userspace one provides more features. From a security standpoint, the userspace kernel should be preferred unless there are definite performance requirements.

The **mount** command is used to interactively access NFS filesystems from other systems. It uses the standard syntax (as used to access local filesystems) with a filesystem of type *nfs*. The **showmount** command when run with a servername argument lists the shares available on that server. When run on the server without any arguments it lists the clients mounting filesystems from that host. The **rpcinfo** command can be used to diagnose NFS problems.

Network File System – NFS (2/2)

- Client
  - /etc/fstab
  - mount options
- Server
  - /etc/exports
  - export options
- Configuration
  - security
  - performance
  - reliability

As with any other filesystem, the filesystem should normally be specified in the */etc/fstab* with appropriate options. This will ensure that the system attempts to mount the filesystem at startup. Note that this may cause problems if the NFS server is unavailable during system startup (unless the mount option "soft" is in use, which is not recommended for data integrity reasons). There are a number of supported mount options specifically for NFS filesystems which alter the behaviour and performance of the the mounted filesystem,

- **rsize** and **wsize** specify the data transfer size in bytes.
- **hard** and **soft** specify the behaviour of programs accessing files when the server becomes unavailable. If hard mounting is used, the program will hang until the server becomes available again. If soft mounting is used, the program will hang for a certain period (specified with the timeo option) after which it will continue. It is generally recommended to use hard mounting to avoid data loss.
- **nolock** turns off NFS filelocking support.

Filesystems to be exported from the server are specified in the */etc/exports* file. The syntax is,

```
filesystem1  client(export options) client client ...
filesystem2  client(export options) client client ...
```

Where *filesystem* is a directory on the server that is to be made available for export and client is a host or group of hosts to allow access to the filesystem. Each client may be followed with a set of optional export options,

- **secure** - only allows requests from ports less than 1024 on client systems.
- **rw, ro** - rw specifies that the client has read-write access to the filesystem while ro specifies read-only access.
- **async, sync** - with async, clients will receive a response from the server before a write request has completed which improves performance but may result in data loss if the server crashes before finishing a write.

NFS exports should normally be read-only unless there are specific reasons to provide write access. In such cases, write-access should restricted to the smallest possible filesystem required and to the smallest subset of clients. You may also wish to use the **root_squash** export option which maps NFS requests from root users on clients to the anonymous uid/gid.

You may wish to experiment with the rsize and wsize parameters to improve the performance of filesystem. You may also find it useful to test the kernel server versus the userspace server. It is recommended to always use the sync export option for read-write shares.

Secure shell – SSH (1/2)

- Features
  - remote login
  - remote execution
  - copying
- SSH versions
- Replaces telnet, rsh, rlogin
- Usage:

```
ssh <username>@<remote host>
scp <local file> <username>@<remote host>:<destination>
scp <username>@<remote host>:<destination> <local file>
```

- X11 Forwarding

SSH is a tool for logging into a remote machine, for copying files to a remote machine and for executing commands on a remote machine. It is intended as a secure replacement for older tools such as telnet, rsh and rlogin. SSH provides a secure connection between 2 hosts. It also provides both password-based and key-based logins. There are 2 versions of the SSH Protocol, version 1 and version 2. An SSH server can be configured to support only 1 only, 2 only or both. This may cause connection problems for some clients (especially old SSH v1 only clients).

The basic usage of SSH is as follows,

```
ssh <username>@<remote host>
```

If ssh can connect to remote host, it will prompt you for a password to login (unless you have pre-configured keys for this system). Commands can be executed on the remote host using the following syntax,

```
ssh <username>@<remote host> <command to execute on remote host>
```

You will again be prompted for the password, but rather than connecting you to that machine, the command will be executed and the output returned to you on your current host.

The **scp** command can be used to copy files to and from remote machines using a syntax like the following

```
scp <local file> <username>@<remote host>:<destination for copy>
          to copy a file to a remote system
scp <username>@<remote host>:<destination for copy> <local file>
          to copy a file from a remote system
```

The **-r** option can be used with scp to copy directories and their contents.

X11 Forwarding

If you use the -X option when ssh'ing to another system and you are running X on the local machine (either under Linux or through an environment such as Cygwin on Windows), then any X commands run on the remote system will display their results on your local system. This is not recommended for slow network connections but works well on LANs.

Secure shell – SSH (2/2)

- SSH keys
  - Commands
    ```
    ssh-keygen -t rsa
    ssh-keygen -y
    ```
  - Files
    ```
    $HOME/.ssh/known_hosts
    $HOME/.ssh/id_rsa.pub
    $HOME/.ssh/id_rsa
    $HOME/.ssh/authorized_keys
    ```
- SFTP
- SSH tunnels
- Putty on Windows

Keys

When the SSH server is installed on a server, it generates a set of unique cryptographic keys. When you login to a new SSH system, the client software copies the public key for that server into the file $HOME/.ssh/known_hosts. If this key changes during any future login, the ssh client will warn you and prompt as to whether you wish to continue connecting. The key might change if another machine is masquerading as the original server or if the system software has been reinstalled on the server.

You may also generate your own set of keys, using the ssh-keygen command as follows,

```
ssh-keygen -t rsa
```

This creates a set of keys in $HOME/.ssh/, one called id_rsa and the other called id_rsa.pub. The .pub file is your public key and can be given to others. You should never give your private key (the first file) to anyone. During key generation you will be prompted for an optional pass-phrase. Setting a pass-phrase gives you enhanced security but the simplest usage of keys does not require it. In the simple usage, you can copy your public key from this machine into the file $HOME/.ssh/authorized_keys on other machines. If you subsequently attempt to login to that other machine, the remote system will recognise the key and allow you to login without a password. This provides a secure mechanism for passwordless logins.

SFTP

Some SSH implementations support the sftp command. sftp provides an alternative mode for copying files between systems running SSH. It provides an interface very similar to that used by the ftp command, while retaining the security of standard SSH.

SSH Tunnels

You can use SSH to tunnel other protocols between systems in a secure pipe. A typical scenario would be where you a network resource somewhere that you want to access (lets say a mail server or a webserver). Unfortunately, there are multiple firewalls between you and it. You can use ssh tunnels to access the resource using the following syntax,

```
ssh -L portA:hostA:portB username@hostB
```

which means

open a tunnel between portB on hostB and portA on hostA.

Putty

Putty is a free windows ssh client available from *http://www.chiark.greenend.org.uk/~sgtatham/putty/*

File Transer Protocol (FTP)

- Overview of FTP
  - operation
  - transfer mode
  - anonymous ftp
- Security risks
  - plaintext passwords
  - anonymous upload
  - buffer overflow
- Server recommendations
  - vsftpd
  - detailed logging
  - anonymous only if required

File Transfer Protocol (**FTP**) is one of the original methods used on unix systems to copy files from one system another over the network. It uses a client/server model. When you wish to transfer files, you (the client) connect to a server (the system you want to upload files to or download files from) using the following syntax,

```
ftp <server name>
```

The server responds with prompts for a username and a password. Successful authentication places the user at the ftp prompt,

```
ftp>
```

From here, a user can **ls**, **cd**, **get <file>** or **put <file>**. Multiple files can be sent or received using **mput <files>** or **mget <files>**. It is recommended to always use **bin**ary transfer mode (see file endings). When finished, signal to the server to **close** the connection.

The ftp command sends usernames and passwords in the clear over the network which is insecure.

Anonymous ftp is form of passwordless ftp. You can enable anoymous ftp on a server with a configuration parameter. This allows people to login to the ftp server without a password (using the special username anonymous). Normally configured for download only, this allows people to download files from a location on your server. Anonymous FTP is normally configured such that users are connected to a particular account on the server (user ftp is the convention) from where they can download files (but not from the system in general). It is important that permissions are correctly set in this directory to prevent anything being written (unless this behaviour is desired).

There are a number of possible security problems with FTP, it thus recommended to only run an FTP server if strictly neccesary. Problems arise with the use of plaintext passwords which hostile parties can view as they travel over the network. If anonymous upload is allowed, hostile parties can upload files into hidden directories without the administrator's knowledge. FTP server software has been subject to security holes in the past which have given intruders system access.

If you require an FTP server, run a secure one such as vsftpd and use detailed logging (which is audited periodically). Also review whether you really need an anonymous ftp configuration.

Exercise 9.1 – Network Configuration

1. **Using the ipcalc tool if desired, suggest a scheme for splitting the network 10.1.1.0 into 3 subnets. List the address range for each one and the netmask and broadcast addresses.**
2. **Identify the latency between your system and a neighbour system and compare this to the latency between your system and www.google.com.**
3. **Identify the systems that a packet passes through on the way from your system to www.google.com**
4. **Using dig, check the IP address of www.example.com**
5. **Using dig, check the name for the IP address 86.43.67.77**
6. **Check if the daytime service is enabled on your system. If not, enable it and test the output from it.**

Exercise 9.2 – Network Configuration

7. **Find out what MPM the installed version of Apache is running.**
8. **Find out the location of the DocumentRoot.**
9. **Place a simple html page at this location and verify that you can view this through the webserver.**
10. **Login to a remote system using ssh.**
11. **Copy a file from your system to another system using scp.**
12. **Execute the host command on a remote system using.**
13. **Generate a key for yourself, add it to the remote system and repeat 9-11 without the use of a password.**

Backups

Backups (1/2)

- Introduction
- Backup cost versus reinstallation cost
- Frequency
  - hourly
  - daily
  - weekly
- Types
  - tape/network
  - onsite/offsite
- Media
  - tape
  - CD-R, DVD-R

Overview

Performing backups is one of a system administrator's responsibilities. The type of backups, the frequency of backups and what exactly is backed up is dependent on the systems being maintained. Generally, your decisions will focus on the cost of restoring the system. It may be cheaper to simply reinstall some systems rather than carry the cost of both storing and recovering the system from backup media. In the case of irreplaceable data such as files created by users, it is normally important to back this up as regularly as possible (in terms of cost, you can put a ballpark figure on the costs by asking yourself what it could cost to have users recreate the data they will lose in the event of a server failure).

Frequency

Backups should be conducted as often as possible. The backup interval should be proportional to the frequency with which the data is changed. If some system only generates new data once a week, there is no value in performing backups more often than once a week (ideally, immediately after the system has generated its data). On typical fileservers in businesses, users will generate a lot of data on a daily, or indeed hourly basis. It is recommended to perform at least daily backups in these situations - more often if it is feasible.

Types

Backup techniques span a multitude of different processes and technologies, some of the most common are listed below in increasing order of value (and cost!),

- backup to remote server
- backup to offsite server
- backup to removable media which is stored onsite
- backup to removable media is stored offsite

It may make sense to perform a combination of these where you perform hourly backups to a remote server and weekly backups to removable media.

Media

Backups have traditionally been performed to tapes which have a high capacity and low cost per megabyte. For smaller businesses, media such as CD-R or DVD-R may be as useful although for long-term storage there are questions over optical media viability.

Backups (2/2)

- Software
  - tar
  - kdat
  - amanda
  - bacula
- Testing of backups
- The tar command
  - `tar -c -f <tar file name> <files/directories to package>`
  - `tar -x -f <tar file name>`
  - `-v`
  - `-z and -j`

Atlantic LINUX

Software

There is a range of software available for making backups on Linux systems. At the simplest level, you can use the cp or scp commands to copy data elsewhere. The tar command has traditionally been the UNIX backup command - it serves both to place files onto tape or to create archive files which can be easily copied to other systems or media. There are also more sophisticated packages including AMANDA, KDat and Bacula which can be used to perform network backups and provide more user-friendly tools.

Testing

A backup system that does not include regular verification of the backup data has no value. Weekly or monthly testing of backups is vital.

tar

The tar command was originally used to archive filesystems to tapes. It can still be used for this purpose although there are more sophisticated tools for tape backups. It is normally used these days to create tar archives directly on filesystems. A tar archive can consist of multiple files and directories. It is common to use tar files for package sources and other distributable data.

To create a tar file

```
tar -c -f <tar file name> <files/directories to package>
```

To unpack a tar-file

```
tar -x -f <tar file name>
```

When using tar, the packing/unpacking and compression/decompression steps can be integrated with either **gzip** or **bzip2** using the **-z** and **-j** options respectively.

e.g.

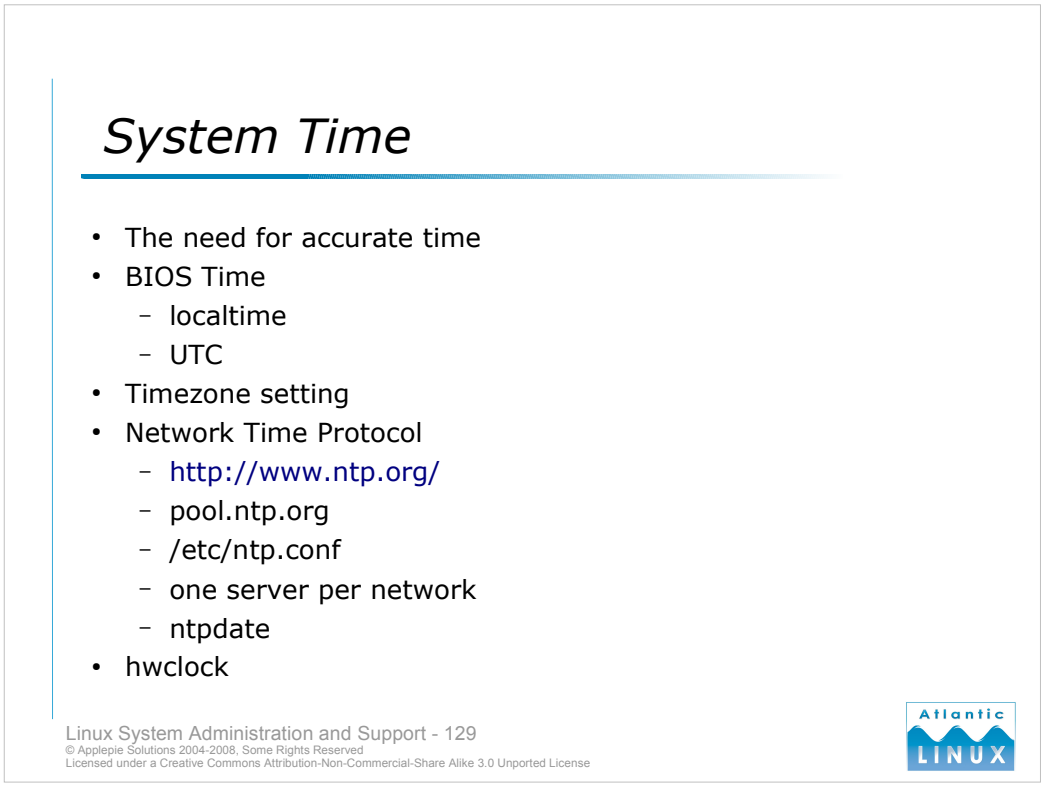
```
tar -zcf <tar file name> <files/directories to package>
```

to create a gzipped tar archive

```
tar -jcf <tar file name> <files/directories to package>
```

to create a bzipped tar archive

System Time



It is important that all of your systems operate with the correct time - ideally to millisecond precision. This is particularly important with systems that use shared storage - you may experience unusual problems with tools such as make if there are time differences between systems using the shared storage. There are a number of steps to configuring accurate time on Linux systems.

BIOS Time

A clock on the motherboard of your system keeps track of the time. On some systems this clock is set to **localtime** and on other systems it is set to **Universal Time Co-ordinated** (**UTC**) which is effectively the same as **GMT**. During installation, your Linux distribution will need to know which format the BIOS time is stored in. This will have a subsequent impact on any time settings. On systems running only Linux, it is recommended to use UTC for the BIOS time. On dual boot systems running Window and Linux, it is recommended to use localtime (this is what Windows expects the BIOS time to be set to). Problems with the system clock changing significantly on a dual boot system between operating systems may be caused by the wrong BIOS time setting. This setting is stored in */etc/default/rcS* on Debian and */etc/sysconfig/clock* on SuSE/NLD systems.

Timezone

You will need to set your timezone during system installation (it can also be subsequently changed in config files listed above).

Network Time Protocol

The clocks on system motherboards are reasonably accurate these days although they may lose accuracy over time and will certainly be incorrect if the CMOS battery on the motherboard fails. Network Time Protocol (NTP) has introduced to allow even systems with unreliable clocks to maintain accurate system time. NTP allows your system to check the time with an NTP server on the Internet which is connected to an extremely accurate atomic clock. Full information on NTP including details of public NTP servers is available from *http://www.ntp.org*. In practice, you simply need to specify an NTP server for your system when installing (these settings are contained in */etc/ntp.conf*). If you have a network of systems, it is considered good etiquette to configure only one of these systems to talk to a public NTP server and to then configure the rest of your systems to talk to this system for accurate time.

NTP is designed to correct small differences between your system's time and the actual time. If there is a large difference in these times, you will need to run the **ntpdate** command to correct it. Most distributions run this command at boot-up to compensate for inaccurate hardware clocks. Most systems also run the **hwclock** command at system shutdown to update the BIOS clock with the more accurate current time retrieved using NTP.

Security

System logs

- syslogd
- /var/log
  - messages
  - syslog
  - apache/
  - ...
- Log rotation
- Reviewing
- dmesg

Linux systems log a lot of information about system activity to a series of log-files under **/var/log**. These log-files can useful when troubleshooting, reviewing system activity or tracking down system intrusions.

System logging is managed by **syslogd** which is configured with */etc/syslog.conf*. This file tells syslogd what messages to log and where to log them.

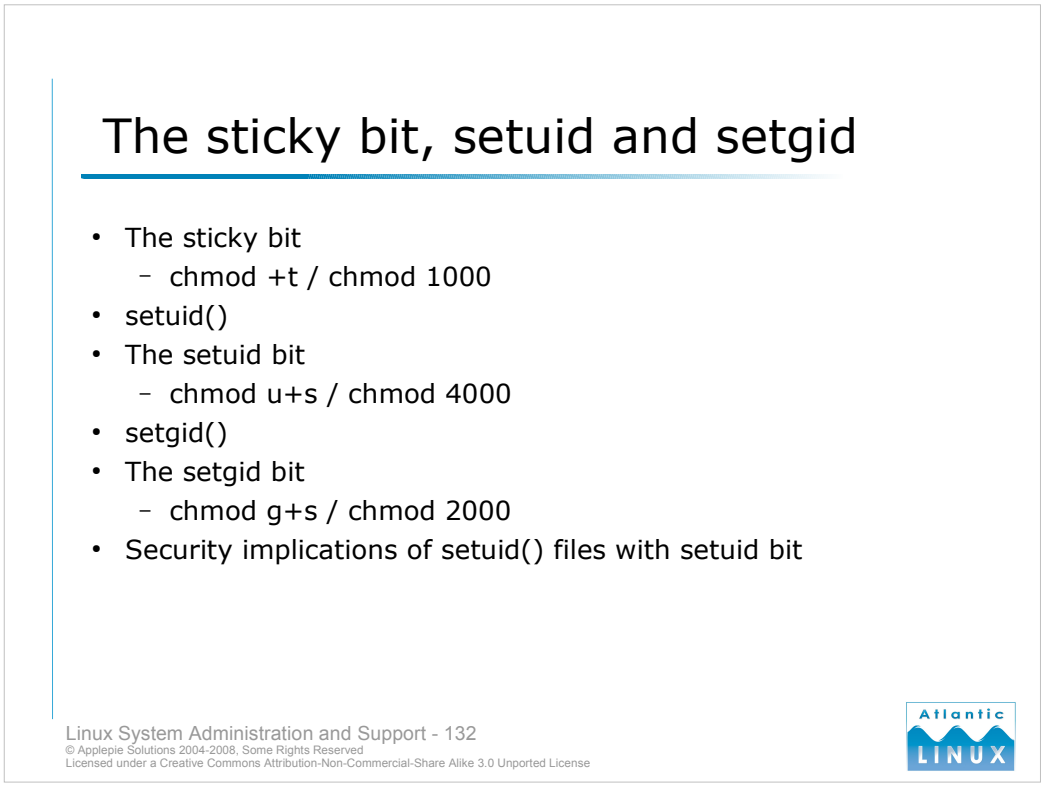
Most Linux distributions log all important messages to **/var/log/messages** and/or **/var/log/syslog**. Additional information is often logged to other log-files under /var/log.

A lot of applications log their activity to separate log-files, applications such as the Apache webserver log their activity to files in either */var/log/apache* or */var/log/httpd* depending on the distribution.

A program called **logrotate** ensures that these log-files are archived and deleted periodically. It can be configured to rotate logs when they reach a certain age or a certain size (without log rotation, the /var/log partition would eventually fill up possibly causing system failures). Logrotate can be reconfigured by editing */etc/logrotate.conf*.

System log-files should be periodically reviewed to identify any problems on the system. Basic tools including grep can be used to scan log-files for particular events or to review the behaviour of specific system daemons. System log-files can be monitored continuously with the **tail -f** command.

The **dmesg** command dumps the current output from the **kernel log buffer** – this usually contains the system boot messages but if there is a lot of kernel activity on the system it may be overwritten by newer messages.



The **chmod** command is used to set file permissions such as whether a file can be read, executed or written to by the owner, group or others. It can also be used to set 3 special permissions.

The sticky bit

Normally, when a directory is group or world-writable on a Linux system, anyone can delete or rename files or directories within that directory. When the sticky bit is set (*chmod +t* or *chmod 1000*) on a directory, only root or the owner of the file file or directory can subsequently rename or delete the file. This is commonly used in directories which are writable by all system users such as /var/tmp and /tmp. World-writable directories should use the sticky bit. The sticky bit has no effect on files.

setuid()

Normally, when a program is run by a user - the program runs with the UID of that user and the GID of that user. Any files created by that program will be created with the UID and GID of that user. The program will not have access to any files that are not acccessible to that user. The setuid() system call allows a program to change its UID when started to any other UID. Only the root user is allowed to call setuid() (to change to a less privileged user).

If the setuid bit is set on an executable file (chmod u+s or chmod 4000) then when the program is executed by any user, it runs with the UID of the user that owns the file. For example, if the executable */usr/bin/foo* is owned by root and has the setuid bit set, when any user runs the program, it runs with the same privileges as root (this normally does not apply for scripts).

The setuid bit and the setuid() system call can be used together. A program owned by root containing a setuid() call, with the setuid bit set can be executed by any user and will run with the privileges of the user specified in the setuid() call.

setgid()

The setgid() system call performs the same function as the setuid() call except it manipulates the GID of the running process. It is also only usable by processes running with root privileges.

If the setgid bit is set on an executable file (*chmod g+s* or *chmod 2000*) then when the program is executed by any user, it runs with the GID of the user that owns the file.

Note:

There are security implications for any programs on that system that are owned by privileged accounts and have the setuid or setgid bits set - you should carefully review any programs (and their source code) before installing them with these privileges.

Password policies

- Policies
  - Disallow simple passwords
  - Require use of numbers and symbols
  - Password Encryption Method
  - Number of significant characters in password
  - Minimum Acceptable Password Length
  - Days to Password Change Warning
  - Days before Password Expires Warning
- Files
  - /etc/login.defs
  - /etc/security and /etc/pam.d
- passwd command

Passwords are the primary way that a Linux system ensures that the user logging into the system is who they claim to be. Password security is thus very important for the protection of the users data, the system and the entire network. It is possible to use alternative mechanisms for verifying user identify but most systems are configured to use passwords which are verified against either a local password file (/etc/passwd and possibly /etc/shadow) or over the network against another server (using a protocol such as YP or LDAP).

You can configure the Linux system to enforce password policies. Where these are configured varies a little between distributions. SuSE/NLD provides the *Security Settings* button on the *Security and Users* screen. Other distributions may require direct editing of files. Most of the settings are stored in */etc/login.defs* with some additional settings being stored in either */etc/pam.d* (on Debian) or in files in the directory */etc/security* (on SuSE/NLD).

Policies

Disallow simple passwords

If enabled, when a user is selecting a new password, the system will reject any passwords consisting solely of a guessable word or a word that is found in the dictionary.

Perform additional checks

If enabled, performs some additional checks on the password such as ensuring it is not too similar to the old one and that it uses a good combination of numbers and symbols.

Password Encryption Method

Specifies the algorithm to use for passwords. Traditionally, the DES algorithm has been used here. This has some limitations including being restricted to passwords of 8 characters. It is also relatively easy for programs with access to the encrypted passwords to guess the plaintext passwords. Alternatives include MD5 and Blowfish, both of which offer better security and should be used if you do not need to interoperate with older UNIX systems password checking.

Number of significant characters in password

This specifies the maximum number of useful characters in a password (limited to 8 in DES), you should not change this unless you are using a different algorithm which supports more.

Minimum Acceptable Password Length

This enforces passwords of a certain minimum length.

Days to Password Change Warning

Allows you to enforce changes of password after a certain period of days.

Days before Password Expires Warning

The user is warned when their password has expired. The system can also lock out accounts which have not changed their password a certain number of days after being warned. This setting controls that behaviour.

TCP wrappers

- Usage
  - inetd
  - libwrap
- Typical services
  - ssh, telnet, finger, ftp, exec, rsh, rlogin, tftp, talk, comsat
- Client request verification
- Access control
  - /etc/hosts.deny
  - /etc/hosts.allow
- Many services implement their own access control (Apache)

TCP Wrappers is intended to provide a level of protection for some of the common services run on a Linux system including - *telnet, finger, ftp, exec, rsh, rlogin, tftp, talk, comsat*. It can be configured to protect a service in 2 different ways, either by linking the service against the **libwrap** shared library or by configuring the service to run using inetd and starting it with the **tcpd** program.

TCP Wrappers performs some logging of client requests and does some basic sanity checking (to confirm that the client is who it claims to be for instance) and then passes the client request to the actual service. It is also possible to configure TCP Wrappers to allow/deny services to/from certain clients on the basis of their address.

TCP Wrappers uses 2 files for access control, */etc/hosts.allow* and */etc/hosts.deny*. TCP Wrappers uses the files in the following sequence, stopping at the first file that matches,

1. Access is granted if a match is found in /etc/hosts.allow
2. Access is denied if a match is found in /etc/hosts.deny
3. Otherwise, access is granted.

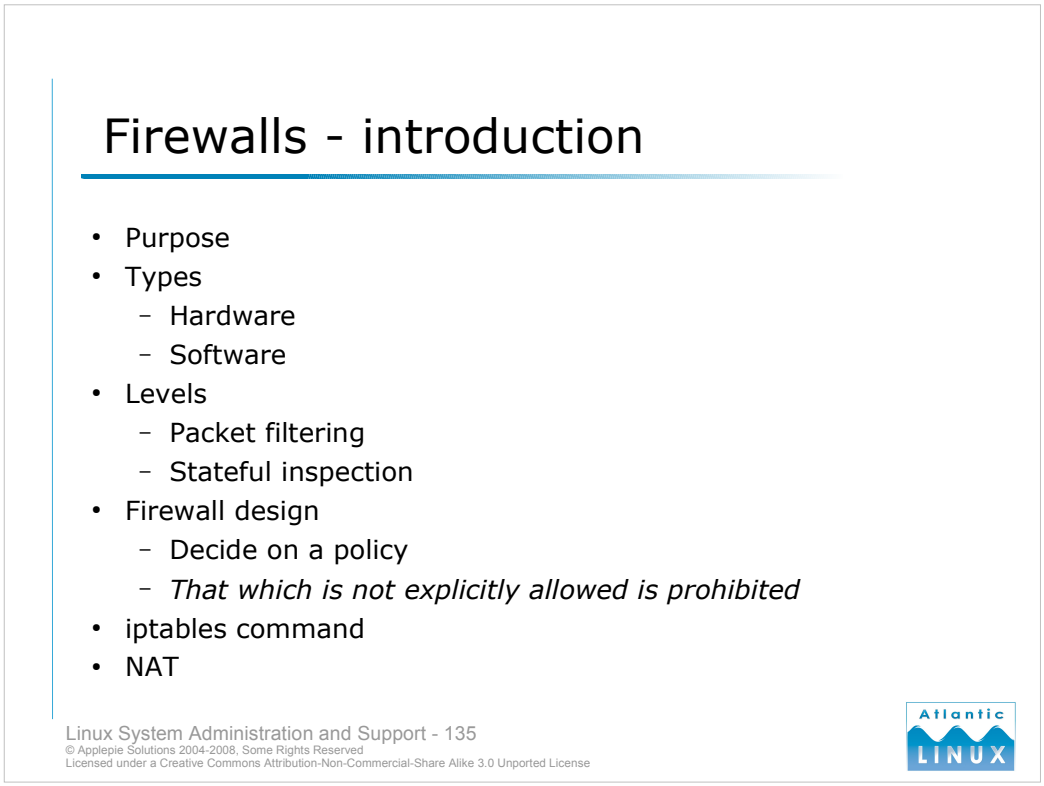
The hosts.allow and hosts.deny files consist of rules which specify one or more services and one or more clients to grant or deny access to, for example

```
/etc/hosts.deny:
          ALL: some.host.name, .some.domain
          ALL EXCEPT in.fingerd: other.host.name, .other.domain
```

The first rule denies some hosts and domains all services; the second rule still permits finger requests from other hosts and domains.

More fine grained control of client access to the system is possible using a firewall where you can control access to particular ports on the system rather than controlling access on the basis of services running behind those ports.

Note that many standalone services such as Apache implement their own access control mechanisms outside of TCP Wrappers.



A firewall is a device used to block unauthorised incoming or outgoing traffic to/from a network. Firewalls can be implemented in hardware, software or a combination of both. They both serve the same purpose. It is common for many hardware firewalls to use Linux "under the hood". Early firewalls provided only packet filtering. This involves the firewall examining the packets and allowing/disallowing the packet on the basis of what type it is. For example, a firewall may deny all incoming FTP packets but allow HTTP packets. More advanced firewalls provide stateful inspection. This involves the firewall examining the packet type, and the state of the packet - so the firewall can recognise the difference between a request for a new connection and a packet which is part of an existing connection. You can thus deny access to new inbound connections while allowing traffic relating to a connection established from within your network.

The first step to implementing a firewall is to design a firewall policy on paper. The policy should describe what types of traffic will be allowed in and under what circumstances. You may also decide to restrict outgoing traffic to particular types (in order to control the effects of a worm or virus on your own network and to restrict what internet services your local network users can avail of). A common starting point for firewall policies is to block all traffic and systematically allow specific types of traffic - summarised as "*That which is not explicitly allowed is prohibited*". This gives much more security by default than an approach which allows everything and then tries to block specific types of traffic. The Linux kernel has included a packet filtering framework since version 1.1. The original packet filtering mechanism was called **ipfw**. This was superceded by the **ipchains** mechanism introduced in version 2.2. This was again superceded in version 2.4 with the **iptables** mechanism which is also found in version 2.6 kernels.

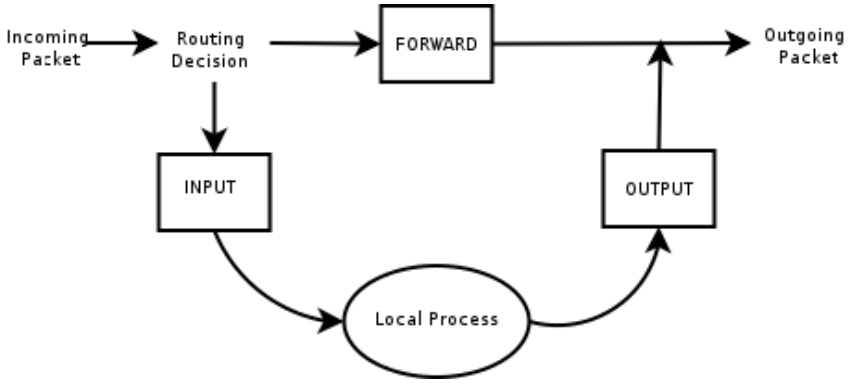
Network Address Translaction (NAT) is often used in conjunction with a firewall. NAT, also known as *network masquerading* or *IP masquerading,* is a technique where the source and/or destination addresses on packets are rewritten as they pass through a firewall. This allows many computers on a private network to access the internet using a single public IP address,

- the firewall translates each private network address to the public IP address when the packet leaves the private network.
- the internet destination for the packet processes the request and responds to the public IP address
- the request is sent back to the firewall which recognises that the packet is intended for a machine on the private network
- the firewall rewrites the destination address of the packet for the private network and sends it on

The iptables mechanism works well with NAT.

iptables - introduction

- Packets entering the kernel are passed through one of the default *firewall chains* (INPUT, OUTPUT and FORWARD)



The iptables mechanism is controlled using the iptables command. Before looking at the command, we need to understand the basic mechanism used by iptables.

The Linux kernel network stack sees each packet received by the system. For incoming packets, the kernel makes a routing decision to decide whether the packet is being forwarded to another host (if the Linux system is acting as a router) or whether the final destination of the packet is this system. Outgoing packets can also be processed by iptables before departing the system. If iptables is in use, all packets are processed by iptables firewall chains. Firewall chains or chains are tables of rules used by iptables. Each type of traffic is processed by a different table.

- Incoming packets which have have this system as their final destination are processed by the INPUT chain before being passed to a local process.
- Incoming packets which are being routed to another system are processed by the FORWARD chain before being passed on to the destination system.
- Outgoing packets are processed by the OUTPUT chain before being passed on to the destination system.

Each chain consists of a set of rules. Each rules contains some criteria for matching a packet and an action to perform on the packet if the match is successful.

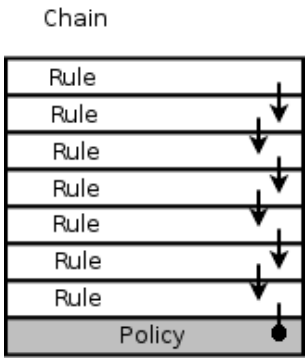
At a higher level again, iptables actually uses a number of tables within which the chains reside. Each table handles a particular type of traffic,

- filter – this is the default packet handler
- nat – this is used for network address translation
- mangle – this is used for altering packet headers

The default table is the filter table and this is the important one from a packet filtering point of view. All of our examples deal with that table.

iptables – firewall chains

- List of rules to be processed
- 3 default chains
  - INPUT
  - OUTPUT
  - FORWARD
- Each packet received by a chain is checked against the rules until a match occurs
- Each rule specifies an action to perform on a matching rule
- If no rules in a chain are matched, chain policy is applied



Iptables uses 3 default chains, INPUT, OUTPUT and FORWARD. A packet being processed by a chain is passed from rule to rule until either a match occurs or it reaches the end of the list of rules. If a match occurs, the target for that rule is applied to the packet and iptables ceases processing it. If a match does not occur, each chain contains a default rule at the end of the chain known as a policy. The policy is applied to all packets which do not match any rule. The order of rules in a chain is thus very important.

The iptables command can be used to modify chains as follows,

iptables -N <chain name> : create a new chain
iptables -X <chain name> : delete a chain
iptables -P <chain name> <target>: set the policy for a chain
iptables -L <chain name>: list the rules in a chain (or all chains if no chain name is specified)
iptables -F <chain name>: Flush (delete) all rules out of a chain
iptables -Z <chain name>: Zero the counters on all rules in a chain

iptables -A <chain name> <rule>: append a new rule to a chain
iptables -I <chain name> <rule number> <rule>: insert a new rule in the chain at a specified position
iptables -R <chain name> <rule number> <rule>: replace the rule at the specified position with the new rule
iptables -D <chain name> <rule>: delete the rule with the specified rule details from the chain
iptables -D <chain name> <rule number>: delete the rule at the specified position from the chain

The iptables command takes many more options including the verbose option (-v), for example, **iptables -L -v** displays a list of current chains and rules along with packet counts for each rule.

Note that chains and rules setup by the iptables command are in memory only and will be lost at next reboot. The **iptables-save** and **iptables-restore** commands can be used to dump the current iptables configuration from memory and load it back into memory at system startup.

iptables – rule matches

- Matches
  - source or destination address
    - single addresses - 192.168.1.1
    - subnets - 192.168.1.0/24
  - protocol
    - udp, tcp, icmp
  - incoming or outgoing interface
    - INPUT rules should only reference incoming interface
    - OUTPUT rules should only reference outgoing interface
    - FORWARD rules can reference both
- ! for inverted rules
- + as wildcard

An iptables rule consists of a match pattern and a target for the packet if it matches. Matches can be made against a wide range of different packet characteristics,

source ip address: -s <ip address>
destination ip address: -d <ip address>

IP addresses can be specified as either single addresses such as 192.168.1.1 or as subnetworks such as 192.168.1.0/24 or 192.168.1.0/255.255.255.0. The notation 0/0 is shorthand for all networks.

protocol: -p <protocol name>

The protocol name can be either a name for the commonly used protocols of tcp, udp and icmp, a number for less common protocols (see the list in /etc/protocols) or the string all to specify all protocols.

incoming network interface: -i <interface name>
outgoing network interface: -o <interface name>

The interface name is the normal Linux device name such as eth0 or ppp0. You can use the special notation + at the end of the interface name to specify all interfaces of that type, for example, eth+.

Note that incoming packets only have an incoming interface associated with them, outgoing packets only have an outgoing interface associated with them and forwarded packets have both an incoming and an outgoing interface associated with them.

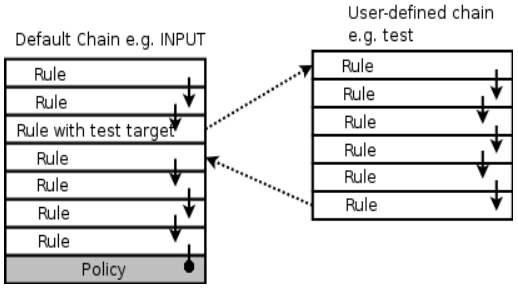
You can also use ! to specify an inverted match, for example,

-i ! eth+

would specify all packets that do not have an eth device as their incoming device.

iptables – rule targets

- Built-in
  - ACCEPT
  - DROP
  - QUEUE
  - RETURN
- User-defined chains



Iptables contains 4 default targets for rules,

ACCEPT - lets the packet through (either to a local process or to another system).
DROP - "drops the packet on the floor" without any further processing.
QUEUE - used to pass the packet to an application on the system for processing, not normally required.
RETURN - used in user-defined chains to pass the packet back to the parent chain for further processing.

User-defined chains
You may also define you own chains for use within iptables. You may do this for organisational purposes within your iptables rules or to avoid duplicating rules which you wish to use with 2 different default chains (such as INPUT and FORWARD). A packet is passed to a user-defined chain by specifying this as the action for a rule in a default chain. If there is no match in the user-defined chain, the packet will be passed back to the next rule in the default chain.

iptables – extensions

- New matches
  - tcp
  - udp
  - icmp
  - mac
  - limit
  - owner
  - state (NEW, ESTABLISHED, RELATED, INVALID)
- New targets
  - LOG
  - REJECT

You can add additional extensions to the core iptables. Extensions can be used to add additional matches and additional targets to iptables. There are a number of extensions commonly bundled with iptables.

Match extensions

tcp

Allows matching against source and desintation ports (such as HTTP) and against certain packet types for TCP traffic.

udp

Allows matching against source and desintation ports (such as DNS) for UDP traffic.

icmp

This allows checking against the type of icmp packet (such as the icmp request sent by the ping command).

mac

Can be used to match rules against the MAC address of the physical ethernet device.

limit

This can be used to limit the number of matches a rule makes in a certain period (useful in conjunction with logging to prevent excessive logging of the same type of traffic).

owner

This can be used to perform matching against the UID and GID of the process that generated the packet.

Target extensions

LOG

This allows iptable rule activity to be logged.

REJECT

This is like DROP but also sends an error to the sending system (rather than silently discarding the packet as DROP dows).

There are many more extensions documented in the iptables man page.

iptables – a simple example

```
# Insert connection-tracking modules (not needed if built into kernel).
insmod ip_conntrack
insmod ip_conntrack_ftp

# Create chain which blocks new connections, except if coming from inside.
iptables -N block
iptables -A block -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A block -m state --state NEW -i ! eth0 -j ACCEPT
iptables -A block -j DROP

# Jump to that chain from INPUT and FORWARD chains.
iptables -A INPUT -j block
iptables -A FORWARD -j block
```

iptables – final notes

- Some alternatives
  - Yast
  - Shorewall
  - Firestarter
  - Guarddog / Watchdog / Guidedog
  - Firewall Builder
- Testing
  - nmap tool
- Resources

It is not neccesary to configure iptables directly. There are a number of pieces of software available for creating and managing iptables rules using either a higher-level language or a gui.

SuSe/NLD comes with a yast2 module under the *Security and Users* screen called *Firewall*. This offers a graphical tool for creating a high-level iptables configuration that will be sufficient for most basic networks.

http://www.shorewall.net/
The shorewall package is also quite popular as an alternative to iptables. It does not provide a graphical user interface. It allows you to describe your network and what traffic you want to allow/disallow in more human-readable terms than that allowed by iptables. Shorewall in turn, takes care of generating the required iptables rules. It is a good compromise between detailed knowledge of iptables and using a simple graphical tool.

http://www.fs-security.com/
The firestarter package provides a graphical tool for creating and maintaining your firewall configuration.

http://www.simonzone.com/software/
guarddog, watchdog and guidedog are a set of graphical tools for KDE for creating and monitoring an iptables firewall.

http://www.fwbuilder.org/
Firewall Builder is a graphical tool for creating firewall configurations for many different software firewalls including iptables. It allows you to generate firewall configurations that can easily be ported to other systems.

Any time you make any changes to your firewall, you should test these using a tool such as **nmap**. This allows you to check what ports are open. **You should run nmap both locally and remotely** (since your firewall will normally treat local traffic differently to remote traffic).

These slides are intended only as an introduction - there are a large number of resources which provide tutorials and further information on iptables. Good starting points are the iptables / netfilter homepage - *http://www.netfilter.org/* and the iptables man page.

Pluggable Authentication Modules - PAM

- History
- Users
  - ssh
  - xdm
  - login
- Authentication schemes
  - /etc/passwd
  - smart card
  - kerberos
  - biometrics
- Linux PAM
- Other PAM implementations

Historically, users logging in to UNIX systems have supplied a password which was verified against the password file before the user was allowed to login. Programs such as telnet, ftp and xdm have used the same file to perform the same verification. Over time, alternative schemes for authenticating users have involved including network authentication schemes and schemes based around using hardware devices such as smartcards. Traditionally, introducing these to a system would have required any users of the authentication mechanism to be rewritten.

Pluggable Authentication Modules (PAM) was introduced to address this problem. PAM-aware programs such as ssh, telnet and ftp can be configured to use any of a number of different authentication mechanisms which can be changed through modifications to a configuration file rather than recompilation. This allows system administrators to customise the authentication mechanisms in use on their system to suit their needs.

It also allows new technologies such as biometric (retina scanning or fingerprint verification) mechanisms to be easily introduced into the system and used by all programs that typically require passwords (ranging from login programs to programs such as xscreensaver).

The *Linux PAM* project is responsible for the implementation of PAM found on Linux. There are also implementations of PAM used on commercial operating systems such as HP-UX and Solaris. These implementations are not entirely compatible with Linux PAM.

Security Advisories

- General
  - CERT - http://www.cert.org/advisories/
- Distribution-specific
  - Novell - http://www.novell.com/linux/security/advisories.html
  - Debian - http://www.debian.org/security/
  - Red Hat - https://www.redhat.com/security/updates/
- SANS Top 20 - http://www.sans.org/top20/

Security problems and new forms of attack are discovered on a daily basis on the internet. Many of these will affect Linux systems you are managing. Most Linux distributions regularly release software updates to rectify these problems. Sometimes, distributions may recommend disabling a particular service or reconfiguring it to close a security hole while a proper patch is prepared.

At a general level - the Computer Emergency Reponse Team in the US (http://www.cert.org/) keep track of new problems and post notices of these problems to allow system administrators to address the problem on their system. These notices are described as advisories and the problems are described as vulnerabilities.

Each distribution also makes available a list of advisories and details of patches that address the vulnerabilities as follows,

Novell - http://www.novell.com/linux/security/advisories.html
Debian - http://www.debian.org/security/
Red Hat - https://www.redhat.com/security/updates/

Note that all 3 distributions are pretty good at releasing critical updates but there may sometimes be a delay. For high security installations, it may be neccesary to disable the piece of software or install a version directly from the author if/until an update is forthcoming from the distribution.

Exercise 10.1 – Security

1. **Attempt to login to your system with an incorrect password and find if this is logged in /var/log.**
2. **Compile the code in the notes as setuid-tester.c (change NEW_UID to another valid UID on your system) and perform the following exercises with the compiled binary,**
   - **a) Run this as yourself and check the ownership of /var/tmp/setuid.tester. Explain if the setuid() succeeded and why/why not. Explain what UID and GID are associated with /var/tmp/setuid.tester and why.**
   - **b) Run this as root and check the ownership of /var/tmp/setuid.tester. Explain if the setuid() succeeded and why/why not. Explain what UID and GID are associated with /var/tmp/setuid.tester and why.**

Atlantic LINUX

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define NEW_UID 1001
#define FILE "/var/tmp/setuid.tester"

/*
 * A short c program for demonstrating the setuid permissions bit
 * and the setuid() system call
 *
 */
int main(void) {
  printf("setuid to %d ", NEW_UID);
  fflush(NULL);
  if ( setuid(NEW_UID) < 0 ) {
    perror("failed");
  } else {
    printf("succeeded\n");
  }

  printf("creation of file %s ", FILE);
  fflush(NULL);
  int fd = open(FILE,O_CREAT|O_EXCL,0700);
  if ( fd < 0 ) {
    perror("failed");
  } else {
    printf("succeeded\n");
    close(fd);
  }
}
```

Exercise 10.2 – Security

c) Set the setuid bit on this file and run this as yourself and check the ownership of /var/tmp/setuid.tester. Explain if the setuid() succeeded and why/why not. Explain what UID and GID are associated with /var/tmp/setuid.tester and why.

d) Set the setuid bit on this file and change the owner of the file to root and run this as yourself and check the ownership of /var/tmp/setuid.tester. Explain if the setuid() succeeded and why/why not. Explain what UID and GID are associated with /var/tmp/setuid.tester and why.

3. Change the *password policy* setting to use MD5 instead of DES.

4. Change the *days to password change* setting to 15 days.

5. Change your user account such that you must specify a new password when you next login.

Exercise 10.3 – Security

6. **Suggest a firewall policy for a small company with 3 PCs and a router connected to the internet and the local PCs; including where the firewall will run, what traffic will be allowed in and what traffic will be allowed out.**
7. **Suggest a firewall rule (in iptables syntax) to drop all traffic from 192.168.1.1.**
8. **Suggest a firewall rule (in iptables syntax) to accept all traffic from 192.168.1.2.**
9. **Suggest a firewall rule (in iptables syntax) to allow only HTTP traffic from machines in the 192.168.x.x network.**
10. **Check what known vulnerabilities exist in the version of Linux running on your system, suggest a strategy for fixing each one.**

Basic troubleshooting

Where to start troubleshooting problems on Linux

1. Check the simple things first.
2. Review any changes recently made.
3. Review known problems for that release.
4. Don't make any assumptions.
5. For remote support, verify all information.
6. Establish a baseline when the system is working.
7. The symptom of the problem may be unrelated to the cause of the problem.
8. Follow a logical sequence of steps.

Boot problems (boot disks and recovery disks, LILO failures)

- System won't boot
  1. Is there media in the drive?
  2. Was a boot-loader installed?
  3. Have drives been moved?
  4. Did the system crash/fail?
- Panic mounting /
  1. Is the correct root specified?
  2. Is the filesystem type correct?
  3. Have drives been moved?
  4. Can you see / with a rescue disk?
- Are the partition tables corrupted?

Filesystem corruption

1. What is the filesystem type?
2. How severe is the corruption?
3. Have you run fsck?
4. Is there important data on the filesystem?
5. Are there backups?
6. Advanced techniques
   - strings
   - photorec
   - dd
7. Recovery tips
   - mount read-only
   - Do not boot windows

Lost passwords

- Do you have an open root session?
- Rescue CD
  1. boot
  2. mount partition with /etc on it
  3. edit /etc/passwd and blank the password field
- Physical removal of drive
- 3rd party rescue CDs
  1. Tomsrtbt - http://www.toms.net/rb/
  2. KNOPPIX - http://www.knopper.net/knoppix/

Printing problems

1. Verify network is working.
2. Verify printer is working.
3. Verify that user has access to printer.
4. Check Windows access.

The Linux kernel

Overview

- Kernel functions
  - Interface to hardware
  - Manage processes
  - Scheduling
- Portability
- Vanilla kernels versus distribution kernels
- Versioning
  - 2.4.x
  - 2.6.x.x
- Kernel modules
- Kernel threads

The Linux kernel is at the core of any Linux distribution - it is the part of the operating system that provides access to the underlying hardware. It is also responsible for scheduling processes (ensuring each program that wants to run is allocated some time to run). The Linux kernel was created in 1991 and released as version 0.01. The current version of the Linux kernel is 2.6 and includes many of the same features found in commercial operating systems such as Windows XP. The Linux kernel is very portable – it currently runs on 15 different types of processor.

Linus Torvalds and a team of other kernel developers continue to work on the Linux kernel source code, developing new versions and releasing them to the world at large from http://www.kernel.org/. Anyone can download these sources, compile it and install the resulting kernel on their system. These kernels are often labelled vanilla kernels. Each Linux distribution also distributes their own version of the kernel. The distribution kernels are created by taking a vanilla kernel and adding additional software patches to the kernel. In some cases, the distribution kernel will be very similar to the vanilla kernel. In other cases, the distribution kernel will be significantly different (in the case of 2.4 kernels distributed by Red Hat, they contained significant functionality backported from the 2.6 kernel series before 2.6 was officially released).

The Linux kernel version is split into numbers A.B.C where A is described as the kernel version, B the major version and C the minor version. The kernel version is only changed when major changes in the kernel occured (it was moved from 0 to 1 when the first official kernel for i386 machines was released and from 1 to 2 when SMP support was added). The major version is used to indicate new releases of the kernel when new functionality is introduced. Even numbers are used for stable releases of the kernel and uneven numbers are used for development versions of the kernel. Up the 2.6, the minor version was incremented any time a new driver, feature or a bugfix was introduced into the kernel. For 2.6, this has changed - the minor number is incremented when a new driver or feature is introduced. For bugfixes, a 4th number - A.B.C.D has been introduced, which is incremented specifically for bug-fixes only.

So 2.6.8 is a stable release of the kernel. 2.6.8.1 is a stable release with some bug-fixes. 2.6.9 is a development release of the kernel. 2.6.9.1 is a development release of the kernel with some bug-fixes. In general, production systems should run the kernels supplied with the distribution.

The Linux kernel can load additional code at runtime (to provide additional device drivers for instance) using loadable modules. Internally, the kernel runs a number of threads which handle different tasks.

Loadable kernel modules

- Purpose
  - device drivers
  - filesystem drivers
- Performance
- Commands
  - lsmod
  - insmod and rmmod
  - modprobe
  - depmod
- /lib/modules
- module versioning
- binary modules

Linux loadable kernel modules are used to add functionality such as device driver or a filesystem driver to a running kernel. This reduces the size of the base kernel in memory since only required device drivers will be loaded. This allows kernels to be distributed which contain support for many different types of device - with only the required functionality being loaded at runtime.

There is no performance impact from using loadable kernel modules - code in a module will be invoked as fast as code in the base kernel once the module has been loaded. Historically, people have recompiled Linux kernels for their particular hardware, removing any additional drivers and functionality not required - this approach is no longer necessary or recommended.

There are a number of commands for managing loadable kernel modules - The **lsmod** command can be used to list out the kernel modules currently loaded. When run, it displays the module name, the size of the module in memory and a list of other modules which are using this module. Modules can be manually loaded and unloaded with the **insmod** and **rmmod** commands. Note that modules often depend on other modules without which they cannot load - so the alsa sound driver *snd* depends on a specific hardware driver such as *snd_intel8x0*. If using insmod, you will need to manually load the required modules in the correct order (and specify the full path to the module). Most distributions run a command at runtime called **depmod** - this scans each module and builds up a table of dependencies (normally stored in a file called *modules.dep* in the modules directory). The **modprobe** command is an alternative command for loading modules - it uses the information from the modules.dep to automatically load any other required modules.

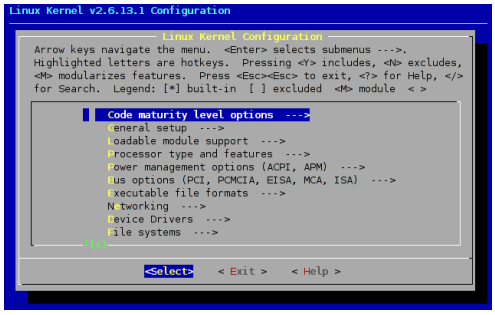
Modules are usually stored under */lib/modules/<kernel version>*. modules.dep can also be found at the top of this directory. The various tools automatically load modules from the correct versioned directory - there can be multiple kernel versions installed on the system at the same time.

When a kernel module is built, it includes information on what version of the Linux kernel source it was built from (this version information can be viewed using the **modinfo** tool). You should only use kernel modules with the same kernel version that they were built from - since they are part of the kernel they have tight dependencies on kernel internals which change from release to release.

Some hardware vendors release Linux device drivers for their hardware as loadable kernel modules. They typically do not provide source code for the drivers. These kernel modules will be tied to a particular version or set of versions of the Linux kernel and may cause problems when used with different kernel versions or even patched releases of the correct version.

Building custom kernels (1/2)

- http://www.kernel.org/
- `tar jxvf linux-2.6.13.1.tar.bz2`
- Software requirements
  - `Documentation/Changes`
- /usr/src
- Configuration
  - `make oldconfig`
  - `make config`
  - `make menuconfig`
  - `make xconfig`



Linux kernels can be downloaded from *http://www.kernel.org/*. You should not need to download and compile a kernel during normal day to day administration. You may find yourself needing to do so to support a particularly new piece of hardware which is not supported in your distribution kernel yet or to compile a customised kernel for a platform with unusual configuration requirements. The current releases are listed on the front page - note that you can download either a patch or the full kernel source. Unless you have an existing kernel source, you probably want the full kernel source file. Note that older releases of the kernel are available from http://www.kernel.org/pub/linux/kernel/ including kernels from v1.0 onwards.

When downloaded, you should extract the kernel source somewhere - the convention is to extract it in */usr/src*. You will normally use the tar command as follows,

```
tar jxvf linux-2.6.13.1.tar.bz2
```

This will create a directory containing the extracted source code. Current kernel sources require about 240MB of disk space before compile and about 450MB after compile depending on what options you compile with. When the kernel source has been extracted, you should cd to the source directory,

```
cd linux-2.6.13.1
```

You will require some software and particular versions to successfully build a new kernel. An up-to-date list of these requirements is in *Documentation/Changes*. Verify that you have the required versions - most modern distributions should come with the right versions although you may need to install some additional packages.

You now need to configure the kernel build options. There are a number of ways to do this. If you want to replicate the configuration of an existing installed kernel, you should copy this configuration file into the kernel build directory (a copy of the running kernel configuration is normally stored */boot/config-<kernel version>*. Copy this to the source directory, renaming it to .config. You now need to run

```
make oldconfig
```

which will check for any new configuration options which aren't in the existing config file and prompt you for values for these. You can also start a new configuration by running any of the following

`make config` - which starts a text based configuration program
`make menuconfig` - which starts a text menu based configuration program
`make xconfig` - which starts a full graphical configuration program

Building custom kernels (2/2)

- Build
  - `make`
- Install loadable modules
  - `make modules_install`
  - `/lib/modules/<kernel ver>`
- Install kernel and related files
  - `cp arch/`uname -i`/bzImage /boot/vmlinuz-<kernel ver>`
  - `cp System.map /boot/System.map-<kernel ver>`
  - `cp .config /boot/config-<kernel ver>`
- Build initial ram disk
  - `mkinitrd -k vmlinux-<kernel ver> -i initrd-<kernel ver>`

    or
  - `mkinitrd /boot/initrd-<kernel ver> <kernel ver>`
- Configure boot-loader with new kernel

Atlantic LINUX

After completing the configuration and saving it, you are ready to build a new kernel with the following command,

```
make
```

If this is not the first time you have built a kernel in this directory, you should run `make clean` before running make in order to clean up any old compiled code.

The kernel build itself may take some time (anywhere from minutes to hours depending on the system you are compiling it on and the number of options enabled). When it has completed, a new kernel should be available in *arch/`uname -i`/boot* (where *`uname -i`* is *i386* for Intel-type systems).

If you opted to build any kernel components as modules, you need to install them on your system using

```
make modules_install
```

which copies the newly built modules to */lib/modules/<new kernel version>*

After compiling, you now need to copy the kernel, System.Map and config files to the appropriate location on the filesystem, normally /boot,

```
cp arch/`uname -i`/bzImage /boot/vmlinuz-<kernel ver>
cp System.map /boot/System.map-<kernel ver>
cp .config /boot/config-<kernel ver>
```

Some Linux systems rely on an **initial ram disk** or **initrd** during the boot-up process. This will be explained in detail later on, for now, note that you will need to create this image at this point for these systems using some variant of the mkinitrd command such as,

```
mkinitrd /boot/initrd-<kernel ver> <kernel ver>
```

(on Debian)

```
mkinitrd -k vmlinux-<kernel ver> -i initrd-<kernel ver>
```

(on SuSE/NLD)

Finally, you must add the new kernel (and initrd image if required) to the boot-loader configuration,

```
vi /boot/grub/menu.lst
```

or

```
vi /etc/lilo.conf
```

And add the appropriate lines to the configuration file (copy an existing entry and use that as a base for your new entry).

Kernel patches

- How patches are used
- diff
- patch
- A typical patch:

```
--- a/drivers/pci/setup-bus.c
+++ b/drivers/pci/setup-bus.c
@@ -40,7 +40,7 @@
  * FIXME: IO should be max 256 bytes.  However, since we may
  * have a P2P bridge below a cardbus bridge, we need 4K.
  */
-#define CARDBUS_IO_SIZE         (256)
+#define CARDBUS_IO_SIZE         (4*1024)
 #define CARDBUS_MEM_SIZE        (32*1024*1024)

 static void __devinit
```


Source code patches are files containing lists of changes to make to one or more files including new lines to add, and lines to remove from the existing file. They are generated using a tool such as **diff**.

Since the kernel source is very large, it is common for developers to make new features and changes available as patches. This allows people to use them without downloading an entire new kernel source (90% of which may be the same as the previous version). This also allows sites such as *http://www.kernel.org/* to distribute both full sources and patches which will allow you to turn a copy of the previous version's source into the current version (in theory - you could take the linux kernel version 1.0 source and apply each successive patch to arrive at the current Linux kernel release).

You will rarely need to apply patches unless you are trying to add functionality not available in either the vanilla kernel or the distribution kernel. Note that vendors normally apply a large number of patches to the vanilla kernel source to generate their distribution kernel. You may occasionally find it useful to take one or more of these patches yourself and apply them to a vanilla source tree.

If you do need to apply a patch, you use the **patch** command as follows,

```
patch -p0 <patch file name>
```

Note that patches are normally applied to a specific kernel version and that it should be vanilla source and not distribution source (which may be patched already) unless specifically indicated by the patch author. It may also be necessary to change directory to the subdirectory containing the file depending on how the patch was generated, looking at the first line of the file should give a hint about this.

Tuning the kernel

- /proc/sys
  - echo 1 > /proc/sys/net/ipv4/ip_forward
- sysctl
  - /etc/sysctl.conf
  - sysctl -A
- Subsystems
  - dev
  - fs
  - kernel
  - net
  - sunrpc
  - vm
  - net

It is possible to tune various kernel setting at runtime. The /proc filesystem includes a special section under /proc/sys which allows settings to be modified. This area consists of a series of directories, subdirectories and files. The files can have values written to them (numbers in some cases, strings in other cases) which change some aspect of kernel behaviour. You can do this in a number of ways. The simplest is from the command-line itself, for example,

echo 1 > /proc/sys/net/ipv4/ip_forward

Turns on a feature in kernel networking called *IP Forwarding*. You can also use */proc/sys* to read the current setting for any of these variables. Note that any changes made under /proc/sys only apply until the system is next rebooted. To have them applied at every boot, you will need to either write a custom script or use the */etc/sysctl.conf* file.

Be careful about making changes to /proc/sys - it is possible to write values to these files which may have a negative affect on the system.

An alternative to directly accessing /proc/sys is to use the **sysctl** command. The command performs the same function as setting /proc/sys directly but provides a more user-friendly interface and does some validation of settings passed to it. It can also be used to list out all available settings. At system boot, most distributions run a copy of the sysctl command which applies any settings found in */etc/sysctl.conf*. This provides a method of storing kernel settings between boots. On some distributions, some other files may also be read at boot-time to set various kernel parameters (*/etc/network/options* on Debian for instance).

The kernel subsystems that can be tweaked are,

- **dev**: settings for specific devices in the system (cdrom devices for example).
- **fs**: tuning parameters relating to filesystems and files.
- **kernel**: global kernel settings and some miscellaneous settings (the behaviour of the kernel when the CTRL-ALT-DEL key combination is placed for example).
- **net**: settings which affect the networking subsystem - documented in more detail in the kernel source in *Documentation/networking/ip-sysctl.txt*
- **sunrpc**: settings relating to NFS
- **vm**: settings relating to virtual memory management.

Initial ram disk - initrd

1) The boot loader leads the kernel and the initial RAM disk specified with the initrd boot parameter.
2) The kernel copies the initial RAM disk image into memory as a normal RAM disk.
3) The kernel mounts the initial RAM disk as /
4) The kernel runs a script called /linuxrc by convention with root privileges.
5) The linuxrc script mounts the normal root file system (using
6) The linuxrc script places the root file system at the root directory using the pivot_root system call (the initrd root remains accessible)
7) The usual boot sequence (e.g. invocation of /sbin/init) is performed on the new root file system
8) The initrd file system is removed

The introduction of kernel modules introduced a problem to the kernel boot process. In order to successfully boot, the kernel needs to be able to access the hard-drive and various other system components. With the introduction of loadable modules, the kernel no longer neccesary includes support for particular types of hard-drive or other system components. The initial RAM disk feature attempts to circumvent this problem by allowing a boot loader load a RAM disk image which can immediately be mounted as the root filesystem. This initial RAM disk can contain a large collection of loadable kernel modules which support any possible hard-drives encountered. These modules can then be loaded by the kernel allowing it to continue to a full-boot, the initial RAM disk root is replaced with the normal system root at this point.

Exercise 11 – The Kernel

1. **Download an extract a copy of the latest Linux kernel source.**
2. **Configure up the extracted kernel with some options (using your existing kernel config as a base).**
3. **Compile this kernel.**
4. **Install the kernel.**
5. **Restart your system using the new kernel.**
6. **List the modules running on the system.**
7. **Load the ntfs module into the system.**
8. **Check for any messages logged.**
9. **Remove the ntfs module from the running kernel.**

Atlantic LINUX

Scripting

Introduction

- When to use shell scripts
  - system maintenance
  - batch operations
  - easily automated repetitive tasks
- When not to use them
  - webserver scripts (CGI)
  - high security jobs
  - heavily loaded systems
- Alternatives
  - Perl
  - Python

Shell scripts are a convenient tool for various system administration tasks. They are generally well suited to specific small tasks. As a general rule of thumb, if a shell script is exceeding a page or two, you should consider re-writing it in more sophisticated scripting language (like Perl or Python).

Shell scripts can be quickly put together and prototyped. Since they are interpreted they will always be slower than a compiled program (but are probably fast enough for most tasks). Large shell scripts can present a maintenance problem.

Do use scripts to automate system management tasks.

Don't use shell scripts on systems or for tasks where security is important – scripts are general not very secure. Remember also that anyone that can run a script can view the contents of that script so passwords or other secrets should never be stored in scripts (the same probably applies to most other interpreted and compiled languages).

Your first shell script

- #!/bin/sh
- Comments
- External commands
- Shell builtins
  - alias, bg, cd, echo, ...
- Shell constructs
  - if, for, while, ...
- Execute permissions

A shell script is just a collection of commands which you want to execute together. More complex shell scripts can use some of the programming language type constructs offered by shells.

The first line of any shell script needs to start with the **shebang - #!*<path to shell>*** this is a piece of *magic* used by Linux to figure out how to execute the script. A bourne shell script will start with **#!/bin/sh**, a bash shell script will start with **#!/bin/bash** and so on. without the #!, you would need to execute the script by passing it as an argument to the sh or bash invoked directly e.g.

```
# sh script.sh
```

Since bourne shell programming is the most portable, the rest of notes and examples in this section will use bourne shell syntax. More powerful shell scripts that are slightly less portable can be created using bash. Shell scripts can be created with any of the other shells including tcsh but it is not generally recommended.

The # symbol is used for **comments**. On any given line, anything after a # is ignored and treated as a comment.

Any valid Linux command is usable within a shell script. The most portable shell scripts will only use basic commands which are guaranteed to be on practically all Linux systems. It is also generally recommended to use absolute paths to commands rather than rely on a users PATH being set correctly (alternatively, you can explicitly set a PATH at the start of the script).

In order to execute a shell script, it needs to have the execute permission bit(s) set. Unlike compiled programs, shell scripts must also have the read permission bit(s) set for any user that wishes to execute the script.

hello worlds

```
#!/bin/sh
# this is a hello world script
echo “hello world”
```

```
#!/bin/sh
# this is another hello world
# script
/bin/echo “hello world”
```

Running a script

- #!
- Running scripts through the shell command
- Debugging scripts
  - -x
  - -v

Executable text-files with #!<shell> as their first line can be executed from the command-line. In addition, any shell script (with or without a #!) can be executed by passing it as an argument to a shell command e.g.

```
# bash script.sh
```

Debugging is also possible with shell scripts by invoking the shell with the **-x** or **-v** options (either in the #! line or by call the shell directly).

-v displays each line of the shell script as it is being run by the shell

-x displays each line of the shell and any arguments as they are being executed. The -x output is generally the most useful for debugging script

Example

```
#!/bin/sh -x
if [ date > /dev/null ]
then
  echo "the date command works!"
else
  echo "the date command doesn't work."
fi

# ./foo.sh
+ '[' date ']'
+ echo 'the date command works!'
the date command works!
```

Shell variables

- Setting
- Using
- Rules for naming
- All variables of type string

Variables an be used in shell scripts the same way they are used in interactive shells.

Variables are set using the same syntax as at the prompt
i.e.

```
VARIABLE=value
```

e.g.

```
foo=10
```

Be careful not to put spaces on either side of the "=". If the value needs to contain spaces, use quotes (") around the entire value.

Variables can be accessed by preceding the variable name with the $ symbol.
e.g.

```
echo $foo
```

would display

10

Valid variable names start with an alphanumeric character or the _ symbol followed by zero or more alphanumeric characters (do not use symbols such as $, ? or * in variable names). Variable names are case sensitive so *$foo* is not the same as *$FOO*.

Shell variables & quoting

- Single quote - '
- Double quote - "
- Back quote - `

Atlantic LINUX

When setting variables in scripts (and at the command-line), it may sometimes be necessary to quote the value as explained previously. The different quote character cause some different behaviors.

When a value is surrounded by **single quotes** (**'**), you are explicitly telling the shell to **not manipulate the string between the quotes**. e.g.

```
# foo='this is the value of foo'
# echo $foo
this is the value of foo
```

When a value is surrounded by **double quote** (**"**) the shell will **expand any variables or backslash sequences** in the string between the quotes e.g.

```
# foo='this is the value of foo'
# bar="this is $foo"
# echo $bar
this is this is the value of foo
```

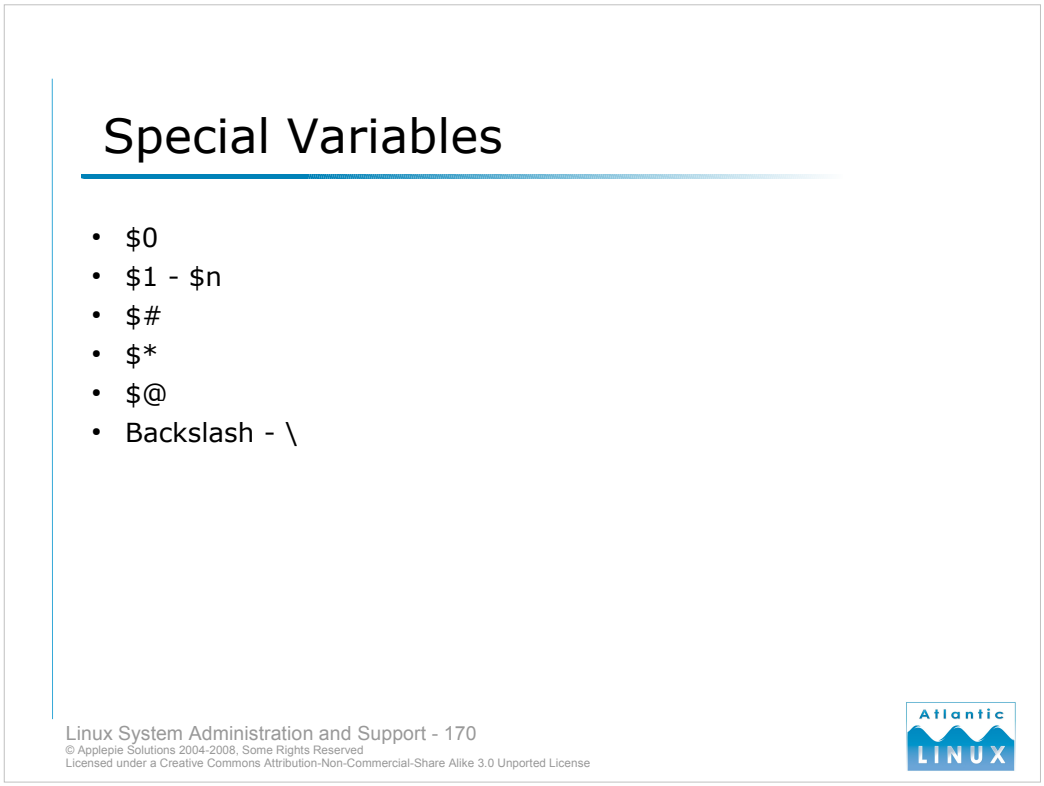
When a value is surrounded by **back quotes** (`) the shell will **attempt to execute the string** between the back quotes and replace the string with the output of the command. e.g.

```
# foo="the time is `date`"
# echo $foo
the time is Thu Nov  4 23:02:57 GMT 2004
```

There can sometimes be a performance improvement from using back quotes rather than other methods, contrast the time taken to run the following:

```
# time find /etc -name *.conf -exec grep foo {} \;

# time grep foo `find /etc -name *.conf`
```



As well as the standard special shell variables (such as $HOME, $PATH, $SHELL and so on), shell scripting in particular also uses a few special variables to handle arguments to shell scripts.

$0 always contains the full path to the shell script itself.

$1-$n contain the arguments to the script (some versions of sh may restrict n to 9 although bash doesn't have this limit).

$# contains the total number of arguments passed to the script

$*** and **$@ contain the same information in slightly different formats. The both contain the values for all arguments to the script. **$*** is a single string of all these values i.e. "$1 $2 $3 $4 ..." while **$@** is a list of all the values in double quotes i.e. "$1" "$2" "$3" ...

The **backslash character -** \ is also known as the **escape character** and is used to instruct the shell or a command to treat the character immediately after it as a normal character, even if it is normally a special character to the shell e.g.

```
# echo \$0
# touch \\foo
# mkdir this\ is\ a\ directory
```

Loops

- for
- while
- until

The bourne shell supports variations of the loop constructs found in most modern programming languages. A common problem in loop constructs and shell scripts in general is missing whitespace where the shell is expecting it.

In the **for loop**, a list of strings is passed to the for loop and some action is performed on each of these items in the body of the loop.

```
for VARIABLE in LIST
do
        BODY
done
```

Newer versions of bash also supports a version of the for loop very similar to that used in the C programming language (but this won't be very portable).

The **while loop** continuously executes BODY while some EXPRESSION is true

```
while EXPRESSION
do
        BODY
done
```

The **until loop** is the opposite of the while loop, it executes the body of the loop until the expression is true.

```
until EXPRESSION
do
        BODY
done
```

The if statement

- if
- else
- elif
- fi

The **if** statement is used to test one or more conditions, the basic syntax is

```
if CONDITION
then
        BODY
fi
```

More complex if statements can be constructed with a second alternative

```
if CONDITION
then
        BODY
else
        BODY
fi
```

Or with many options

```
if CONDITION
then
        BODY
elif CONDITION
then
        BODY
else
        BODY
fi
```

case and test

- case
- test
  - STRING1 = STRING2
  - STRING1 != STRING2
  - INTEGER1 -eq INTEGER2
  - INTEGER1 -ne INTEGER2
  - -f FILE
  - ...

The **case** statement is an alternative to a chain of ifs and elifs. The syntax is very similar to that used in the C programming language and others

```
case EXPRESSION in
          ( PATTERN )
                BODY
          ;;
          ( PATTERN )
                BODY
          ;;
esac
```

In the case statement, if the EXPRESSION matches any of the PATTERNs, the related BODY is executed. It is common to use * for the last PATTERN to make a default action

A lot of these constructs rely on a true or false value to decide what action to take. Every command returns an EXIT CODE which can be used as a true/false value (exit codes are covered later on). We can also use the **test** command to test various conditions including string and integer equality, numerical comparison, the states of files (whether they exist, are a specific type, have a particular size and so on). the normal syntax is **test *expression***. An alternative syntax uses **[]**'s i.e. **[*expression*]**

When testing variables that may not have a value, it is recommended to use the following expression to avoid failures in test due to a missing variable,

```
if [ ${foo}X = "valueX" ]
```

where *value* is the expected string in *${foo}* and *X* is used to protect the test.

Exit codes,functions

- Exit status
  - checking
  - exit
  - $?
- ${VARIABLE}
- Functions
  - arguments
  - returning status

Every Linux command returns an *exit status* or *return code*. By convention, this code is 0 when the command has been successful and a non-zero value specific to the command when the command fails (in the range of 1-255). Scripts can use this both to test the return codes of commands they invoke (at the simplest level, a 0 return code is treated as true by test) and to return their own status code (using **exit <status code>**). The return code from the previously executed command is also stored in the special shell variable **$?**.

A significant error in shell scripts is failing to check $? after executing a command the output from which the script depends on to complete.

When referencing variables in shell scripts, it is sometimes important to be able to embed the variable in a string. When doing this, you need to indicate to the shell interpreter where the variable begins and ends. This can be achieved by surrounding the variable name with {}'s - $VARIABLE is the same as ${VARIABLE} so you can create scripts like,

```
echo "Results are stored in ${USER}_RESULTS.data"
```

Shell scripts can also be written to use rudimentary functions or procedures. The basic syntax is

```
FUNCTION_NAME()
{
        BODY
}
```

A function must be defined before it can be invoked (there are no function prototypes in shell scripts). Functions can also take parameters which use the same variables as shell script arguments ($1 .. $9). They can also return a status using the shell construct **return**.

Special devices

- Useful special devices
  - /dev/null
  - /dev/zero
  - /dev/random and /dev/urandom

The Linux filesystem contains a number of **special devices** which can be useful for shell scripting.

The **/dev/null** device is a data sink, any data written to /dev/null is discarded by the system. It can be useful in scripts to discard out the stdout or stderr of a command when executing it e.g.

```
#!/bin/sh
#
if [ date > /dev/null ]
then
  echo "the date command works!"
else
  echo "the date command doesn't work"
fi
```

The **/dev/zero** device is a source of *\0* (also know as NUL) characters. It can be used to feed dummy data to commands e.g

```
$ cat /dev/zero > zeros.txt
```

The **/dev/random** and **/dev/urandom** are a source of random data which be more useful than zeroes in some cases. The system generates random data using various sources of entropy on the system including network traffic and so on. The /dev/random device will only return random data if there is enough **entropy** on the system. The /dev/urandom device will always return random data but it may not be as random in low entropy situations.

sed & awk

- Both perform text transformations
- Operate on standard input (from a pipeline) or files
- Can also be used to build scripts in their own right
- Come in different flavours
- Support regular expressions

sed 's/regexp/replacement text/{flags}'
e.g. sed 's/o/_/g' foo.txt

awk -F <chars> ' { print $n } ' filename
e.g. awk -F, '{print $3, $2, $1}' csv.txt

Two other commands commonly used in shell scripts and **one-liners** are the commands **sed** and **awk**. Both commands can actually be used as scripting languages in their own right or can be incorporated into shell scripts.

They are powerful tools for transforming text in various ways (substitutions, re-arranging files and so on) and can take their input from standard input, a command pipeline or a file.

They support complex syntax and have a lot of overlapping functionality but a common use of sed is for performing text substitutions while awk is often used for selecting particular columns from text data.

Exercise 12 – Scripting

1. **Write a script which, when run in a directory, renames all files in that directory to an all lowercase version of the original filename.**
2. **Write a script which takes and processes the following options. The script should display an error message when the arguments to the script are incorrect,**
   - **-h displays a help message**
   - **-l performs an ls (with each line preceded by the command name)**
   - **-d displays the date in the form "22:34 25-Dec-2004"**

In closing ...

- Summary
- Next steps
- Questionaire

Thank you and well done!