Linux Fundamentals

Institut Pasteur Tunis
21 March 2007

History and Copyright

John M. Ostrowick, jon@cs.wits.ac.za
School of Computer Science,
University of the Witwatersrand
June 2005

Heikki Lehväslaiho, heikki@sanbi.ac.za
SANBI, University of Western Cape
March 2007

Contents

Session 1 Overview

- Introduction to Linux and Linux history
- User interfaces
- Getting started: user authentication
- Desktop environment
- Common GUI applications
- Linux file system and home directories
- Pathing
- File manipulation through the GUI

What is Linux?

- A multitasking, multi-user operating system
- Informally refers to the operating system as well as the standard tools and applications distributed with it
- Specifically, Linux refers to the *kernel* which forms the core of the operating system
- The kernel is distributed with indispensable utilities and applications, such as compilers, editors, command interpreters, etc.
- Most Linux software distributed under the GNU general public license (GPL)

What are Linux Systems Used For?

- Linux is based on Unix operating systems, traditionally associated with
  - heavy computing, stability and backend services
  - computationally intensive tasks such as visualisation and graphics rendering
  - scientific computation and simulations
  - academic laboratories
- Large portion of the Internet is Unix-based
- Linux is revolutionising the old legacy of Unix by bringing the operating system to desktops and everyday users

What are Linux Systems Used For?

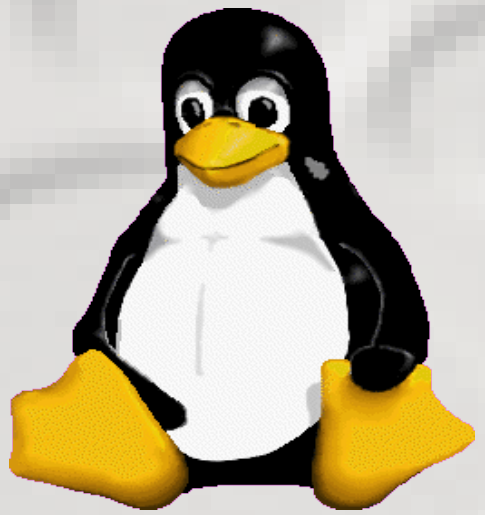
- Linux systems often used for back end services:
  - Web servers, database servers, file servers, mail servers, ftp servers, firewalls, routers, print servers...
- Linux is slowly moving onto the desktop:
  - Desktop, office suites, graphics manipulation
- Growing commercial interest in Linux-based computing:
  - Reliable, secure IT systems
  - Cost-effective solutions
  - Support from traditional Unix companies such as Sun, HP, IBM, Novell

Brief History (I)

- Linux began in 1980's as an effort to create a *free* Unix-like operating system
- The project was called *GNU* and was run by the Free Software Foundation (FSF) created by Richard Stallman
- Development began with system tools such as editors, a compiler and hundreds of other utilities
- By early 1990's most of the components were written, but the operating system was missing a kernel
- Coincidentally, Linus Torvalds of Helsinki University had been working on a Unix-based kernel – the first version was completed in 1994

Brief History (II)

- Linus liked the endeavours of the Free Software Foundation and released his kernel under the GNU GPL
- The Linux kernel and GNU tools made a complete, free operating system: the GNU/Linux operating system



Open Source Licenses

- GPL was one of the most important contributions of the FSF
- The Open Source definition (http://www.opensource.org) is based on the GPL
- Open Source licenses ensure basic freedoms, including:
  - The freedom to use the software for any purpose
  - The freedom to distribute the software to others
  - The freedom to modify the software
  - The freedom to distribute the modified software to others (under the same licensing conditions)
- GPL, MPL and BSD licenses are some examples

User Interface

- Describes the way a system interacts with its users
- Text-based or command line interface:
  - Dates back to pre 1980's
  - Commands typed using keyboard to run applications
  - Less user-friendly but extremely flexible, especially for system administration
- Graphical interface:
  - Point and click to run applications
  - Interaction with system easier and quicker to learn
- Linux provides both and can be set up to boot in either text mode or graphical mode

Logging In

- Since Linux is a multiuser operating system, users must authenticate themselves before gaining access
- Authentication is done with a username and password, configured by the system administrator
- Although visually different, the process of logging in the same in both text and graphical mode
- The combination of username, password and disk space for personal files is called a user *account*
- **Note** that Linux is case-sensitive

Switching Between Text and Graphics

- When booting in text mode, the desktop is launched using the command *startx*
- When booting in graphical mode, a command interpreter can be launched from the application menu
- The command interpreter is also called a *terminal* or *shell*
- *Ctrl-Alt-F1* to *F6* will switch from graphical mode into a text-based terminal
- *Alt-F7* will switch back to graphical mode if the above step was performed
- *Alt-F1* to*F6* will switch between several text-based terminals

Changing Passwords

- To change your password, type the command *passwd* at a shell
- You will be prompted for a new password, and a confirmation - after confirming your current password
- Bad passwords are disallowed – passwords should be at least 6 characters long, contain both letters and digits or punctuation and must not be based on dictionary words
- There is usually a graphical utility for changing passwords accessible from the application menu (this is desktop-specific)

The Desktop Environment

- A number of different desktops are available for Linux, each with different look & feel, and functionality
- Currently, most popular free desktops are KDE and Gnome
- Both are distributed with the most popular Linux distributions
- Graphical applications may be
  - desktop-specific: e.g. k-tools for KDE
  - non desktop-specific: e.g. OpenOffice, Mozilla

Desktop Features

- Main desktop area:
  - Application windows
  - Shortcut icons
- Panel:
  - Application menu launcher, offering convenient access to commonly-performed tasks
  - Application shortcuts, should be customised according to user's needs
  - Desktop switcher, to switch between virtual desktops, allowing the user to group applications logically without cluttering
  - Taskbar, allowing the user to manage currently running applications
  - System information

Useful Graphical Applications

- Word processing / Spreadsheets / Presentations: OpenOffice.org Writer / Calc / Impress
- Drawing: OpenOffice.org Draw
- Project management: MrProject
- Image manipulation: GIMP
- Web browsing: Mozilla firefox
- Email: Evolution, Mozilla thunderbird
- Text editor: Emacs
- PDF reader: Adobe Acrobat reader, xpdf
- Accounting: Turbocash, gnucash
- IRC client: xchat

KDE-specific Applications

- kedit: simple text editor
- korganizer: calendaring and event organiser
- kghostview: postscript document viewer
- kcalc: scientific calculator
- kpaint: bitmap drawing program
- kmail: graphical email client
- amarok: CD player
- khelpcenter: online help application
- konqueror: file and Web browser
- kword: word processor
- kspread: spreadsheet application

Gnome-specific Applications

- gedit: simple text editor
- ggv: postscript document viewer
- gcalctool: scientific calculator
- nautilus: file and Web browser
- eog: graphics viewing program
- gnumeric: spreadsheet application
- yelp: help browser
- gnomemeeting: Voice over IP suite
- rhythmbox: CD and music player
- gnome-pilot: Palm PDA management

Miscellaneous Utilities

- Screen locking: password enabled screen saver
- Panel configurator: customise look & feel, location, behaviour and shortcuts on panel
- Online help
- Find files utility
- Logout function: to quit the desktop, log out, shutdown or reboot the computer
- Control panel (requires *root* access): to configure hardware, software and system settings

File System Basics (I)

- Files are entities for storing data in a computer system
- There are many types of files: various data files and programs; even devices are represented as files
- Filename extensions are a convenience for the user – the operating system does not derive any meaning from it
- Some common extensions include:
  - .bz2: File zipped with the bzip2 utility
  - .c: C source code file
  - .gif/.jpg/.png: Image files (GIF / JPEG / PNG)
  - .gz: File zipped with the gzip utility

File System Basics (II)

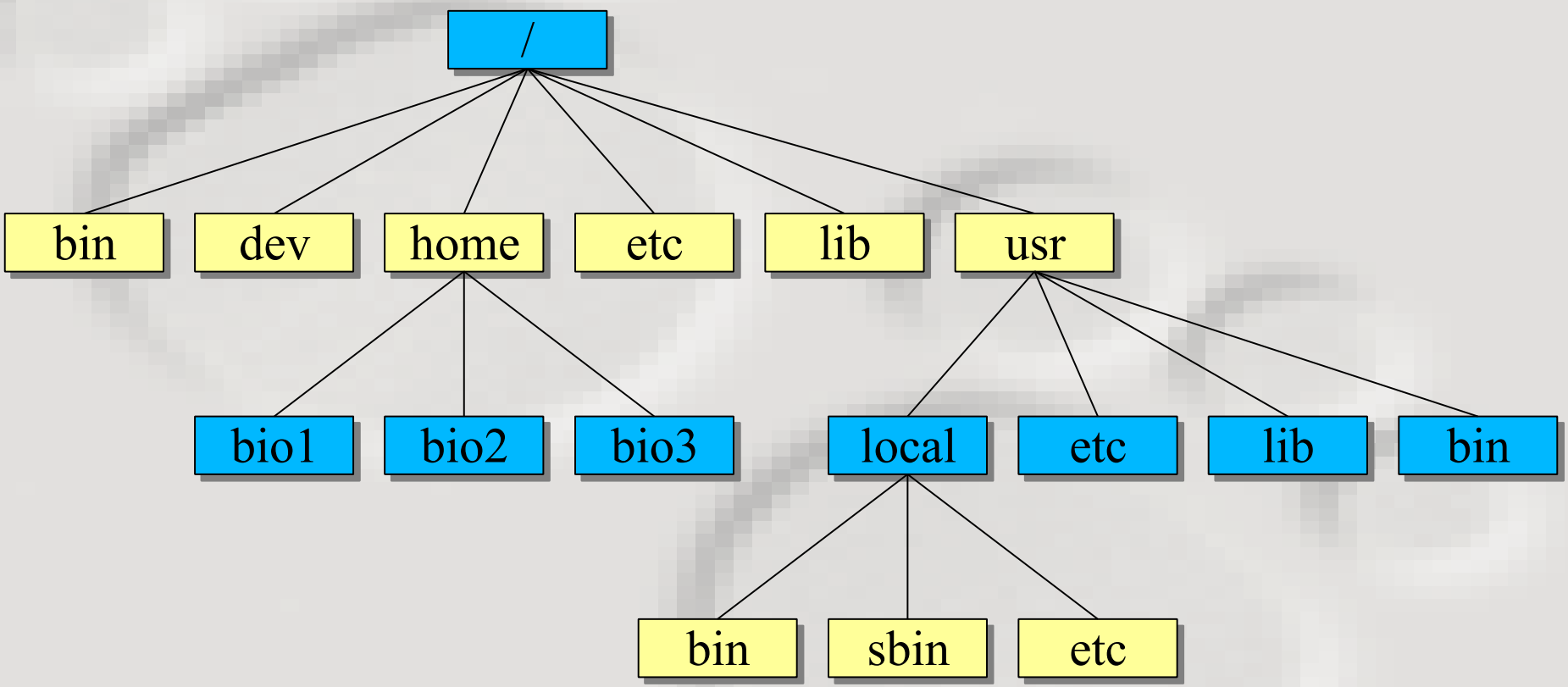
- Common extensions (cont.):
  - .html: Web page
  - .mp3: MP3 audio file
  - .pdf: PDF document format
  - .pl: Perl script
  - .rpm: RedHat software package
  - .odt: OpenOffice.org files (writer / calc / impress / draw)
  - .tar: Archive created with the tar utility
  - .txt: Plain text file
  - .zip: File compressed with the zip utility

Note:
Executables
do not have
a standard
extension

Directory Hierarchy

- Files are grouped into logical units into collections called *directories* (known as folders in other OS's)
- Directories may contain subdirectories, resulting in a hierarchical structure
- The top-most directory in this tree is called the *root* directory, denoted by a /
- Each user has a directory set aside for storing personal files – this is called his *home directory* – uniquely identified by the username e.g /home/dilbert
- Users should create new directories in their home directories to properly organise their files

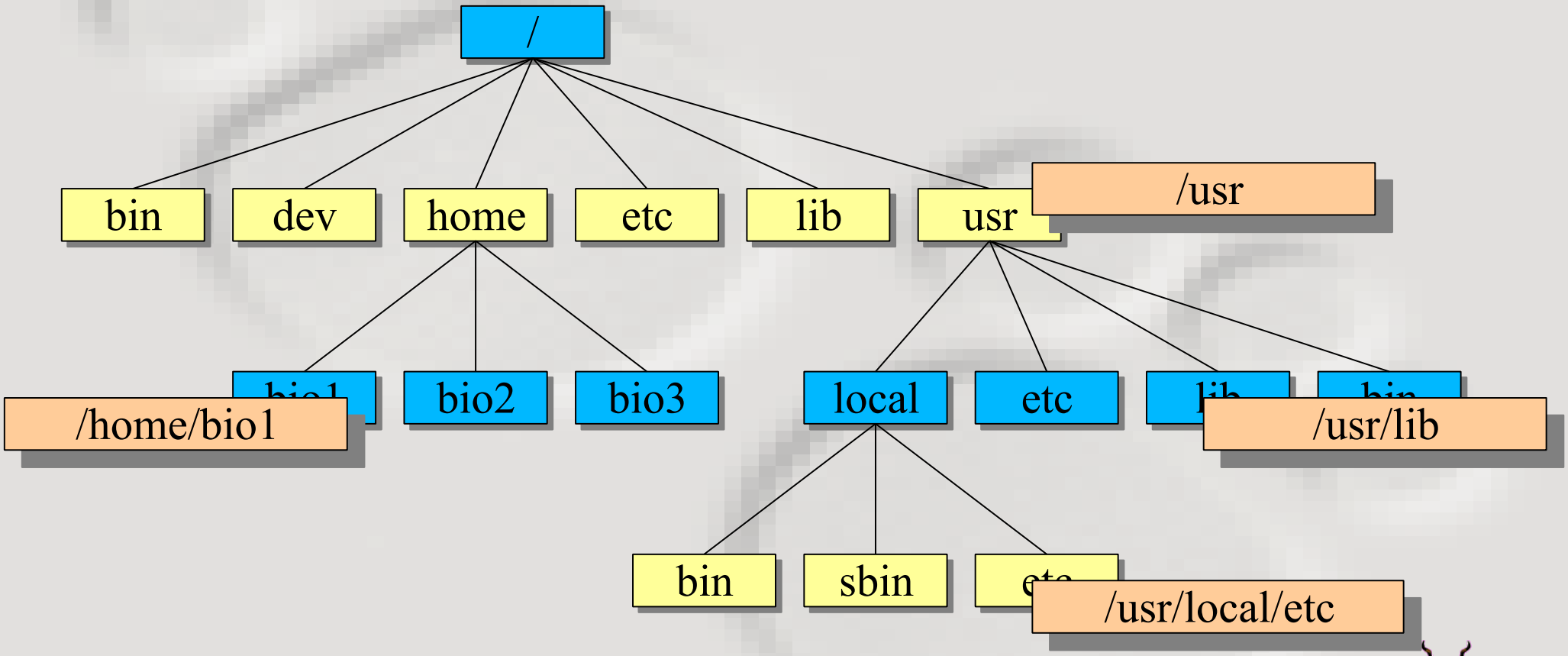
Example Directory Tree



Pathing

- The location of a file in the file system is known as its *pathname*
- For example:
  - /home/dilbert/admin/budget.doc
  - /usr/bin/less
- A pathname uniquely defines the path from the root directory to a file
- Note that applications are also files in the file system and have their own pathnames

Pathing


/
bin
dev
home
etc
lib
usr
/usr
bio1
bio2
bio3
/home/bio1
local
etc
lib
bin
/usr/lib
bin
sbin
etc
/usr/local/etc

File Manipulation with the GUI

- *konqueror* is a KDE utility for visualising and navigating the file system
- The location bar displays the directory whose contents are being displayed
- The main window can be configured to display information in different ways
- Directories and files can be manipulated through menu options, shortcut icons and context-sensitive menus (i.e. by right-clicking on an object)
- File permission information can be accessed through the *properties* option (covered in more detail later)

Session 1 Command Summary

Command	Description
`startx`	start the graphical display
`passwd`	change a user's password

Session 2 Overview

- Command-line interface (CLI)
- File manipulation with the CLI
- Viewing file contents; text editors
- File system security – users and groups
- Shell job control

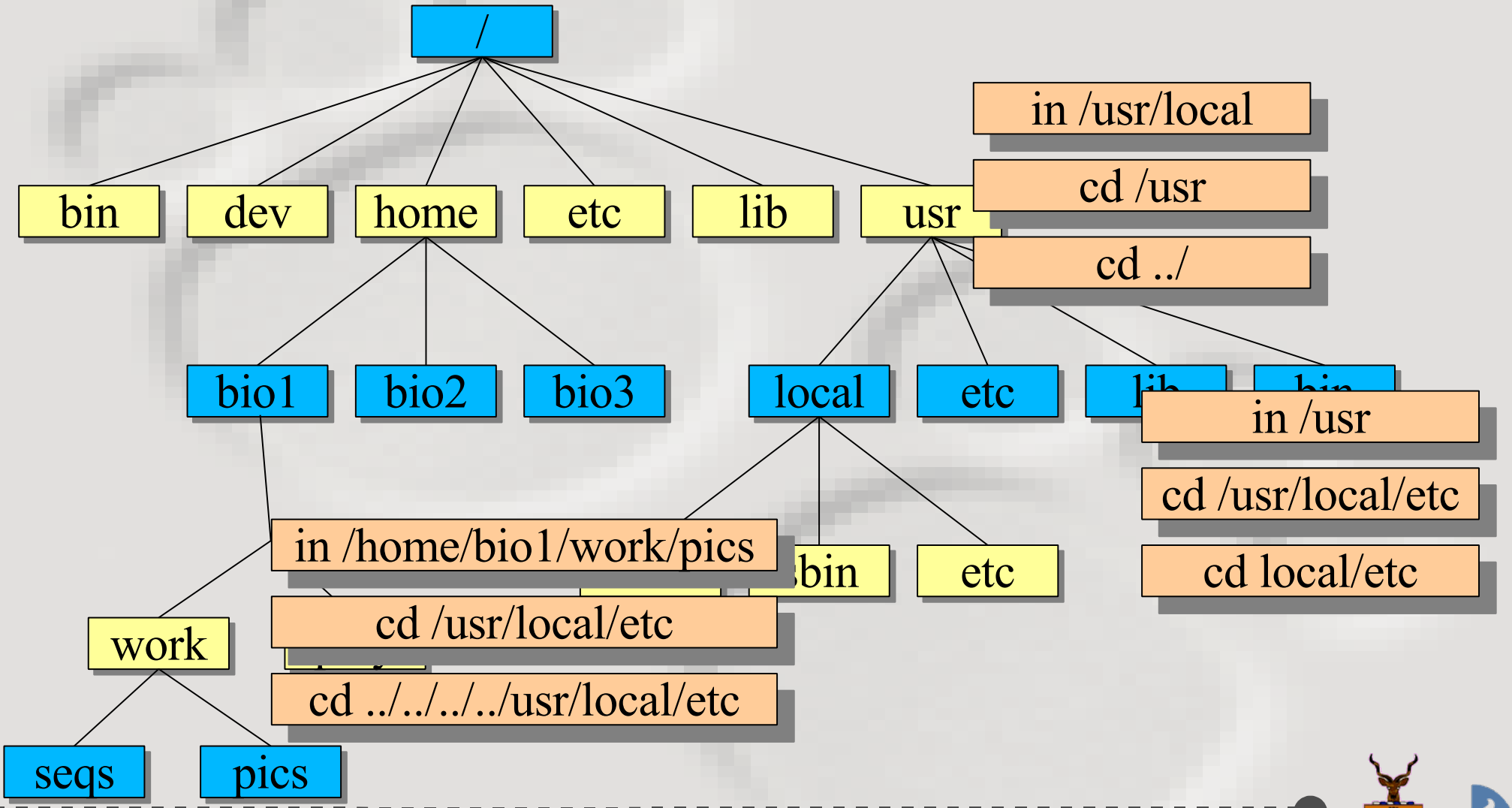
File Manipulation with the CLI

- Understanding paths is important when using the CLI
- *Absolute* pathname: a path that describes the location of the file from the root directory, e.g. /home/dilbert/admin/budget.doc
- *Relative* pathname: a path that described the location of the file from the current directory, e.g. admin/budget.doc
- A user is automatically placed in his home directory when logging in or opening a new terminal or shell
- The command *pwd* prints the current working directory

Changing Directory

- The *cd* command is used to change directory – pathing rules apply, for example
  - cd /home/dilbert/admin
  - cd admin
- Certain symbols have special meanings for directories
  - ~ refers to the user's home directory
  - . (dot) refers to the current directory
  - .. refers to the parent directory
- For example
  - cd ~/admin
  - cd ../../bin

Pathing


/
bin
dev
home
etc
lib
usr
bio1
bio2
bio3
local
etc
work
seqs
pics
in /usr/local
cd /usr
cd ../
in /usr
cd /usr/local/etc
cd local/etc
in /home/bio1/work/pics
cd /usr/local/etc
cd ../../../../usr/local/etc
bin
etc

Command Structure and Options

- Linux commands typically follow the structure

  command [options] argument1 argument2 ...

- Options are shown in square brackets and are just that (optional). Options take the following forms:
  - Single dash followed by a single letter (e.g. -d; -h)
  - Double dash followed by the long name of the option (e.g. --delim; --help)
- Most commands support the *-h* and *--help* options
- Arguments are a mandatory part of the command and must be supplied

Listing Files

- Command: **ls** [options] [files]
- Common options:
  - -a: shows all files, including hidden files
  - -l: uses long listing format
  - -r: produces output in reverse order
  - -t: sorts output by modification times
  - -1: lists one file per line
- Examples:
  - ls (short file listing)
  - ls -al (long listing, including hidden files)
  - ls -1 (short listing; one file per line)
  - ls -lrt (long listing; most recently accessed files last)

Creating & Removing Directories

- To create a directory, use *mkdir <directory>*
  - mkdir admin
  - mkdir /home/dilbert/admin
- To remove a directory, use *rmdir <directory>*. Note that the directory must be empty
  - rmdir admin
- Again the pathing rules apply. The easiest method is to change directory first so that relative pathing can be used

Copying Files

- Command: **cp** [options] source destination
- Common options:
  - -f: does not prompt before removing
  - -i: prompts before removing
  - -r: copies directories recursively
- Multiple files can be specified as the source, but only one destination can be specified (which may be a directory)
- Examples:
  - cp budget.doc oldbudget.doc
  - cp jan-budget.doc feb-budget.doc admin/

Removing Files

- Command: **rm** [options] files
- Common options:
  - -f: does not prompt before removing
  - -i: prompts before removing
  - -r: removes directories recursively
- Examples:
  - rm budget.doc
  - rm budget.doc oldbudget.doc
  - rm -r admin/ (to be used with care!)

Renaming and Moving Files

- Command: **mv** [options] source destination
- Common options:
  - -f: does not prompt before moving
  - -i: prompts before moving
- Multiple files can be specified as the source, but only one destination can be specified
- This command is also used to move and rename directories
- Examples: mv budget.doc oldbudget.doc; mv budget.doc ../admin; mv admin/ admin2003/

Using Wildcards in Filenames

- Wildcards can be used to refer to multiple files
  - * represents any string of characters
  - ? represents a single character
  - [] defined sets or ranges
- Examples:
  - ls *.doc
  - mv *.doc olddocuments/
  - rm *
  - ls -l A???.txt
  - ls [Aa]*png
  - ls [a-z]*jpg

Helpful CLI Features

- Tab completion: command and file names are completed as far as possible when the tab key is pressed. Double-tab key press shows available completions
- History: pressing the up arrow key scrolls backwards through the previous commands
- Events (!): previous events can be rerun using the ! character and the first character(s) of the event. The most recent matching event is chosen. !! runs the most recent command
- Control-R allows live history searching
- These features are shell-dependent (bash supports all)

Viewing File Contents

- *cat* utility: outputs the contents of a file to the terminal
- *less* utility: similar to *cat*, but displays one page of output at a time (improvement of *more*)
  - Use spacebar to advance to the next page
  - Use B to jump back to the previous page
  - Use Enter key to advance line at a time
  - Use up and down arrow keys to move a line at a time
  - search by pressing '/', type the string and press enter (press n for next)
- *clear* utility: clears the screen

Text Editors

- Linux offers a variety of text editors: vi (or vim), emacs, nedit, pico, jed, kwrite, etc.
- *vi* (and vim – vi-improved) is a command-driven editor that is found on almost all Unix-based systems
- *Emacs/xemacs* is a GNU editor that offers a large amount of additional functionality. Its graphical interface and maturity make it an excellent choice of editor for the novice user.

File System Security

- Linux file system security is a simple scheme based on users and groups
- Users belong to one or more groups, set by the system administrator
- Groups allow file access to sets of users to be easily implemented
- Each file is owned by one user and allocated to one group
- A new file is created with the user as its owner and the user's current group as its group
- File ownership can be changed with the *chown* command

Privilege Types

- Files and directories may be granted read, write and execute permissions
- Each of these privileges are specified separately for:
  - the owner
  - the group
  - other users, who do not fall into the previous categories

Privilege Semantics

- Privileges have different meanings for files and directories
- Privileges for files
  - read permission allows the file to be read, copied, printed, etc
  - write permission allows the file to be modified, overwritten and deleted
  - execute permission allows the file to be executed
- Privileges for directories
  - read permission allows the directory's contents to be listed
  - write permission allows files to be created and deleted in it
  - execute permission allows the user to change directory to it

Viewing Permissions via CLI

```
-rw-r--r-- 1 heikki heikki 177932 2007-03-07 13:29 questions.pdf
```

- The *ls -l* command shows file and directory permissions in the first column
  - If the first character is a dash, then it represents a file. If it is a d, it represents a directory
  - Characters 2-4 indicate the permissions of the owner (r = read, w = write, x = execute)
  - Characters 5-7 indicate the permissions of the group
  - Characters 8-10 indicate the permissions of other users
- Third column displays the owner
- Fourth column displays the group

Modifying Permissions via CLI (I)

- Command: **chmod** [options] mode files
- Common options:
  - -R: applies the changes to directories recursively
- Mode specifies:
  - Entities to which the change should apply (u = user, g = group, o = other, a = all)
  - Whether permission should be granted (+) or revoked (-)
  - Permission types that should be granted or revoked: r, w and/or x

Modifying Permissions via CLI (II)

- Examples:
  - chmod g+rw budget.doc (grants read and write access to group)
  - chmod o-rx public_html (revokes read and execute permissions to others)
  - chmod ug+x MakeBudget (grants execute permission to user and group)
  - chmod a+rwx public_html (not a good idea!)

Shell Job Control (I)

- Job control refers to the ability of the shell to run processes in the background
- Background processes do not accept input from the shell, useful for:
  - processes that do not produce any output
  - processes that do not interact with the shell
  - processes that will take a long time to execute
- A background process is assigned a job number

Shell Job Control (II)

- Start a process in the background by appending an ampersand to the command, e.g. *mozilla &*
- Suspend an active processes by keying *Ctrl-Z*
- Send a process to the background by typing ***bg*** *<jobnumber>*
- Send a process to the foreground by typing ***fg*** *<jobnumber>*
- View background and suspended processes with the ***jobs*** command

Session 2 Command Summary

Command	Description
`pwd`	print working directory
`cd`	change directory
`ls`	list files and directories
`mkdir/rmdir`	make / remove directories
`cp`	copy files and directories
`rm`	remove files
`mv`	move / rename files and directories
`cat`	print files to the terminal
`less/more`	filter output for convenient viewing
`clear`	clear the screen
`chown`	change file and directory owner and group
`chmod`	change file and directory access permissions
`fg/bg`	send processes to foreground / background
`jobs`	list background and suspended processes

Session 3 Overview

- IO redirection
- Text processing utilities
- Getting help on commands
- Accessing remote services

IO Redirection

- Many Linux commands take input (STDIN) and / or produce output (STDOUT) on the terminal
- IO redirection allows both input and output to be replaced by files
- Output redirection: The **>** symbol redirects output to a file rather than the terminal
- Input redirection: The **<** symbol redirects input from a file rather than the terminal
- Examples:
  - ls > temp
  - wc -l < temp

IO Redirection: STDERR

- Many Linux commands report to a third default location: standard error, STDERR
- tcsh can not redirect STDERR to a file!
- STDERR redirection in bash:
  - **2>** redirects standard error to a file rather than the terminal
  - **2>&1** redirects standard error to the same file as standard out (equivalent to shorter **&>filename**)
- Examples:
  - prog > temp 2> log
  - prog &> outfile.$$

Pipes

- Pipes redirect the output of one command to the input of another
- This allows the user to combine commands to create more complex ones
- Examples:
  - ls -1 | wc -l
  - cat somefile.txt | grep the
  - who | grep mary | wc -l

Searching Within Files

- Command: **grep** [options] pattern files
- Common options:
  - -c: prints a count of the matching lines instead of the default output
  - -i: performs a case-insensitive search
  - -n: also prints out the line number
  - -v: inverts match, printing out all non-matching lines
- Examples:
  - grep bash /etc/password (search for “bash” in the given file)
  - grep -v the novel.txt (search for any line *not* containing “the”)

Looking at only one end of the file

- Command: **head** [options] file
- Command: **tail** [options] file
  - -n: where n is number of lines to display
- Examples:
  - head file (display 10 first lines)
  - head -210 filename | tail (look at line numbers 200-210)

Differences Between Files

- Command: **diff** [options] file1 file2
- Common options:
  - -i : ignores changes in case
  - -B: ignores changes that just insert or delete blank lines
  - -q: reports only whether the files differ
- Examples:
  - diff newfile.txt oldfile.txt (list differences between the files)
  - diff -i newfile.txt oldfile.txt (list differences with case-insensitive comparison)

Extracting Columns from Files

- Command: **cut** [options] filename
- Common options:
  - -d *delim*: uses the given delimiter, instead of tab
  - -c *range*: outputs only specified characters
  - -f *range*: outputs only specified fields
  - (Range in the form *N*, *N*-, *N*-*M* or -*M*, counting from 1)
- Examples:
  - cut -f1-3 mydata.txt (cut fields 1 to 3, use tab as separator)
  - cut -d”,” -f2 summarydata.csv (cut field 2, use comma as separator)

Merging Files in Columns

- Command: **paste** [options] files
- Common options:
  - -d *list*: uses delimiters from the *list*, instead of tabs
  - -s: pastes one file at a time instead of in parallel
- Examples:
  - paste -d”,” cols1.txt col2.txt
    (paste columns from the 2 files with comma as the separator)

Extracting Rows from Files

- Command: **split** [options] filename
- Common options:
  - -b *size*: outputs *size* **bytes** per file
  - -l *size*: outputs *size* **lines** per file
- Examples:
  - split -l 200 output.db
    (split file into 200 line segments)

Sorting

- Command to sort: **sort [options] file**
- Common options:
  - -f: folds lower case characters to upper case
  - -b: ignores leading blanks
  - -r: reverses the sort
  - -n: numeric sorting
- Examples:
  - sort -rf mydictionary
    (output lines in case-insensitive reverse sorted order)
  - sort -n somefile | uniq
    (output lines in sorted numeric order)

Removing Duplicates and Counting

- Command to remove *successive* identical lines:
  uniq [options] file
- Common options:
  - -c: prefix lines by the number of occurrences
- Examples:
  - sort somefile | uniq
    (output lines in sorted order, removing duplicates)
  - sort somefile | uniq -c | sort -nr
    (count occurrence of lines and show most common first)

Passing program output as arguments

- White space limited list as arguments to an other program:
  xargs **[options] command**
- Common options:
  - -d: set delimiter
- Examples:
  - cut -d: -f1 /etc/passwd | sort | xargs echo
    (compact listing of all logins)
  - ls -t | head | grep .ppt | xargs mv -t w/talks/
    (move the latest ppt files into the w/talks directory)


Getting Help on Commands

- Command: **man** [section] name
- Common options:
  - -k: searches the database for appropriate man page entries
- Standard use displays the manual page of the command
- The section number may need to be specified for keywords that have more than one entry in the system
- Examples:
  - man ls
  - man -k cron
  - man 5 crontab

Remote Access

- Remote access refers to the ability to connect to another machine on a network and work as though physically located at that machine
- Two applications allow a shell to be run on a remote machine: *telnet* (older) and *ssh* (secure shell)
- *ssh* encrypts the traffic between the two machines, and is preferred to *telnet*
- *scp* is a related *ssh* utility that provides secure file transfer, and is preferred to *ftp*

Secure Shell (SSH)

- SSH command

  ssh [-l username] hostname OR

  ssh username@hostname

- SCP command

  scp [[user1]@host1:]file1 [[user2]@host2:]file2

  Arguments provide the source and destination respectively

- Examples:
  - ssh -l root guests.cs.wits.ac.za
  - scp ../docs/budget.doc guests.cs.wits.ac.za:documents/
  - scp guests.cs.wits.ac.za:backup.gz .

Session 3 Command Summary

Command	Description
`grep`	print lines matching a pattern
`diff`	find differences between two files
`cut`	remove sections in columns from files
`paste`	merge files as columns
`split`	split a file into pieces
`sort`	sort lines of text files
`head`	output the first part of the file
`tail`	output the last part of the file
`uniq`	remove duplicate successive lines from a text file
`xargs`	pass list as arguments to an other program
`man`	display online manual pages
`ssh`	secure shell client (remote login program)
`scp`	secure copy (remote file copy program)

Session 4 Overview

- Compression and archiving utilities
- Process management
- Shell concepts
  - Environment variables
  - Aliases
- Scheduling utilities

Compression and Archiving (I)

- Compression and archiving are useful for backups and transferring multiple files across a network (via ftp, http, scp, email attachments, etc.)
- Compression utilities include gzip (.gz extension), bzip2 (.bz2 extension) and zip (.zip extension – MS compatible)
- Archiving utilities include tar (.tar extension – most common Linux format) and zip (.zip extension – MS compatible)

Compression and Archiving (II)

- Command: **gzip** [options] files
- Common options:
  - -d: decompresses instead of compressing
  - -l: lists compression information
  - -t: tests the file's integrity
- Examples:
  - gzip somefile.txt (compresses the file and renames to somefile.txt.gz)
  - gzip -d tarfile.tar.gz (uncompresses the file and renames to tarfile.tar)
- **bzip2** works similarly to **gzip**, with a *.bz2* filename extension

Compression and Archiving (III)

- Command: **tar** [options] [files]
- Common options:
  - -c: creates a new archive
  - -f *tarfile*: uses the specified tar filename (instead of stdin / stdout)
  - -t: lists the contents of an archive
  - -v: lists files as they are processed
  - -x: extracts files from an archive
  - -z: filters the archive through *gzip*
  - -j: filters the archive through *bzip2*

Compression and Archiving (IV)

- Examples:
  - tar -cvf docbackup.tar *.doc (creates a tar file containing all .doc files)
  - tar -zxf somearchive.tar.gz (extracts files in the archive compressed with *gzip*)
  - tar -jtf somearchive.tar.bz2 (lists files in the archive compressed with *bzip2*)

Compression and Archiving (V)

- Command: **zip** [options] zipfile file1 file2 ...
- Common options:
  - -r: recurses subdirectories
  - -T: tests the file's integrity
- Examples:
  - zip jan-budget.zip jan-budget.sxc (creates zipped archive containing the single file *jan-budget.sxc* – note: original file is not modified)
  - zip mail-backup.zip mail/* (creates zipped archive containing everything in the *mail* directory)

Compression and Archiving (VI)

- Command: **unzip** [options] zipfile
- Common options:
  - -d *directory*: specifies the directory to which to extract
  - -l: lists archive contents without extracting
- Examples:
  - unzip -d mail jan-backup.zip (unzips into mail/ directory)
  - unzip -l jan-backup.zip (lists the contents of the archive)

Process Management

- Linux is a multitasking operating systems that allows more than one process to be run at one time
- A running program is called a process; associated with it is a process ID (PID)
- Processes can run in the foreground or background, and can be combined in interesting ways using IO redirection

Viewing Processes (I)

- Command: **ps** [options]
- Common options:
  - -a: shows all processes attached to a terminal including those owned by other users
  - -l: displays additional information
  - -u: displays additional information about the user
  - -w: wide format, not truncated at end of line
  - -x: includes processes not attached to a terminal
  - -U user: filters according to specified user

Viewing Processes (II)

- Examples:
  - ps (list processes in current terminal of current user)
  - ps -aux (list all processes)
- **top** offers similar information, but updates itself continuously

Terminating Processes

- Processes no longer responding can be terminated with the kill command: **kill** [-signal] PID
- This command can be executed at various signal strengths. Signal strength 9 is the most brutal – only use as a last resort
- Common signals are:
  - 2: Interrupt signal (same effect as Ctrl-C)
  - 9: Emergency kill signal: cannot be ignored by a process
- Examples:
  - kill 1964 (kill process with PID 1964 as gently as possible)
  - kill -9 1145 (kill process with PID 1145 using maximum force)

Shells (I)

- A shell is a command interpreter that executes commands entered through the command-line interface
- Several shells are available, most popular are *bash* (Bourne again shell) and *tcsh* (successor of the original C-shell)
- The shell a user uses is set by the system administrator, but can be changed with the *chsh* command

Shells (II)

- Shells mostly offer the same functionality but may differ slightly
  - Different initialisation files (*bash* runs .bashrc and .bash_profile; *tcsh* runs .cshrc)
  - Tab completion
  - possible command / filename completion (*tab* in *bash* vs *Ctrl-D* in *tcsh*)
  - tcsh should not be used fro scripting; can not redirect standard error

Environment Variables

- They define the user environment and are read from initialisation files each time a user logs in
- To view the value of a variable, type *echo $VARNAME* or to see all, type *printenv*
- Some common environment variables:
  - EDITOR: sets the editor to be used by programs such as mail clients
  - PATH: specifies directories to be searched for executables
  - SHELL: the default login shell
- To reload any initialisation file without having to logout and in again, type *source <filename>*
  - e.g. source ~/.bashrc

Some Shell Specifics

- Using bash:
  - Global initialisation file is */etc/profile*
  - User-specific initialisation files are *.bash_profile* and *.bashrc*
  - *set* displays all currently set variables
  - Syntax to set a variable: *export VARNAME="value"*
- Using tcsh:
  - Global initialisation file is */etc/csh.cshrc*
  - User-specific initialisation file is *.cshrc*
  - *setenv* displays all currently set variables
  - Syntax to set a variable: *setenv VARNAME="value*"

The PATH Variable

- Specifies the directories that the shell searches to find a command or executable
- Directories are searched in the order they appear
- Any user-directories added to a path should come after the system directories
- If the current directory is added to the path, it should always be the last entry

Aliases

- Aliases provide command-substitution functionality. They can be used to create new commands or modify the default behaviour of existing commands
- The *alias* command is used to view and create aliases
  - called with no arguments, it prints out the current aliases
  - *alias name=value* creates a new alias
  - custom user aliases are stored in *.bashrc* or *.cshrc*
- Examples:
  - alias rm='rm -i' (change the behaviour of *rm* to confirm deletes)
  - alias ll='ls -lLF | more' (create a new command for friendly file listings)

bash as programming language

- An other way to provide command-substitution functionality is bash functions
- The *set* command is used to view bash functions
  - more versatile than aliases; you can combine any commands
  - *name() = { commands }* creates a new function in *.bashrc*
- Examples:
  - psg() { ps -AF | grep "$@" | grep -v grep ; }
  - killn() { kill `psg "$@" | cut -c9-14` ; }
- Bash is a full featured programming language
  - Advanced Bash-Scripting Guide

Scheduling Utilities

- ***cron***
  - Allows jobs to be scheduled to run at particular times, and is generally used to execute repeated tasks
  - It operates by executing tasks when the system time matches a defined pattern. eg. *cron* can be told to clean up temporary files every Monday at 7am
  - The cron service is started at system startup and then wakes up every minute to check if a job needs to be started
  - The cron is modified with the *crontab* command, *crontab -l* lists
- ***at***
  - *at* is similar to *cron*, but is used to execute once-off tasks, eg. a*t* can be told to run *find* the next time 8:15 rolls around by typing 'at 08:15 -c find'

Editing the Cron

- Use the *crontab -e* command to edit the cron(, or **kcron)**
- Cron jobs are specified using an obscure syntax – type *man 5 crontab* for good documentation
- There are 6 columns in the file specifying the following (an * in the column leaves it unspecified):
  - 1: minute (0-59)
  - 2: hour (0-23)
  - 3: day of month (1-31)
  - 4: month (1-12)
  - 5: day of week (0-7; 0==7==Sunday)
  - 6: the command to be executed

Cron Examples

```
# run 5 minutes after midnight, every day
5 0 * * * $HOME/bin/daily.job >> $HOME/tmp/out 2>&1
# run at 10pm on weekdays, annoy Joe
0 22 * * 1-5 mail joe “Where are your kids?”
# run at 14:15 on the first of every month
15 14 1 * * $HOME/bin/monthly-reports
```

Session 4 Command Summary

Command	Description
gzip/bzip2	compress and decompress files
tar	archiving utility
zip	package and compress files
unzip	extract compressed files in a zip archive
ps	generate process status report
top	display top CPU processes
kill	terminate a process
echo	output text to the terminal
source	read and execute commands from a config file
set	print or set shell variables
export	export variables to the environment
alias	print or set aliases
crontab	maintain crontab files for individual users
at	execute a command at a specified time

Session 5 Overview

- Linux installation process
  - Discussion of various installation options
  - Demonstration and discussion
- Discussion of Linux systems and services

Installing the Software

- Most popular distributions have a graphical installer that offers
  - Step by step instructions
  - Detailed information screens, help and warnings
  - Automated detection and configuration of most hardware
  - Intelligent default options and values
  - Customisation at various levels of granularity: for first-time to expert users

Single versus Dual Booting

- Dual booting allows multiple operating systems to be installed on the same machine
  - Operating system loader allows the user to choose which operating system to load at boot time
  - Useful for home and desktop computers
  - Requires hard drive space to be partitioned before installation to create separate disk space
- Single booting applies when only one operating system is installed
  - Standard choice for server installations

Installation Types

- Some installers offer different installation types
  - Recommended, customised and expert; or
  - Workstation, server and customised
- Inexperienced users should opt for precustomised installations
- Additional software can always be added at a later stage
- Installation disks can also be used for system *upgrades* in which case existing user data is preserved

Disk Partitioning (I)

- Sections the hard drive(s) into different areas
- Useful for keeping data logically separate, e.g. keeping programs away from user data
- A special partition called *swap* is usually created – virtual memory partition as an extension of RAM
- If Linux is installed on a single disk, it is not necessary to partition the disk further
- If keeping the Windows partition, defragment first

Disk Partitioning (II)

- Possible additional partitions include
  - */boot* for kernel files
  - */home* for user home directories
  - */usr* for program files
  - */tmp* for temporary system files
  - */var* for variable sized system data, such as log files

Configuring Hardware

- Most (possibly all) computer hardware will be automatically detected by the installer
- Still a good idea to know the model of hardware components in the computer
- Uncommon and old hardware is not always supported by Linux
- Note that there is sometimes a lag between the release of new hardware and Linux support due to reverse engineering of drivers

Selecting Software

- Most installers will allow you to configure the list of software to be installed, even if a specific installation type has been chosen
- Additional software that you may want includes
  - alternative desktops
  - development packages
  - scientific packages
  - uncommon software
  - Linux services (server applications)

Installing Services

- Services are applications which offer some functionality to other machines, called clients
- Linux systems are incredibly flexible in terms of server-side services they offer
- They can be set up as print, file, Web, mail, news and many other types of servers
- Linux systems are so reliable that often one machine is used to offer a number of different services
- **Note**: Security becomes an important consideration when offering services on a Linux machine – this is beyond the scope of this course

Internet Services

- Web server
  - Manages incoming HTTP requests and serves web pages to clients requesting them
  - Apache is the most popular Linux web server - can be combined with dynamic Web systems such as CGI (Perl) and PHP
- Mail server
  - A mail server manages incoming mail connections for users on the local machine
  - Sendmail and Postfix are popular Linux mail servers

Remote Access Services

- ftp server
  - Facilitates file uploads and downloads from a machine running this service
  - Uses the FTP protocol standard, which means that clients are available for most operating systems
  - Packaged with inetd (collection of simple Internet services)
- ssh daemon
  - The ssh daemon allows remote users to connect to the machine, providing them with a shell on the server
  - Can be used to transfer files, using a “sister” client program called *scp*
  - OpenSSH is the currently used implementation

Database Services

- A number of proprietary databases exist for Linux, such as Oracle, Sybase and Interbase
- In addition, open source offerings exist although these are not as mature
  - PostgreSQL: the most mature open source database, well-supported
  - MySQL: fast, lacks some traditional database functionality, later versions have added them

File Services

- Remote Linux file systems can be seamlessly incorporated into a local file system with the *mount* utility
- Windows file systems are supported through Samba
  - Windows file systems can be imported to the local system
  - Linux file systems can be exported (i.e. made to look like) a Windows drive

Startup Mode

- System can be configured to boot in graphical or text mode
- Graphical mode is a good option for workstations, where graphical applications are mostly used
- Text mode is a good option for servers
  - servers do not usually need a graphical interface
  - reduces system resource needs and increases stability
- Note that it is still possible to change between modes after startup, as well as to change the default startup mode after installation

User Accounts

- Administrative account *root* always created during installation
- The *root* account is used to manage all system configuration such as management of software, services and users
- The root password need to be good and kept secret!
- At least one other non-administrative account should be created, but this can also be done after the installation process
- Some distributions () use *sudo* instead of separate root account that gives password protected full privileges to the first user.

Session 6 Overview

- User management
- Linux file system structure
- File system types
- Mounting devices
- File system utilities

User Accounts and Groups

- Linux is a multiuser operating system, where multiple users can work simultaneously in their own operating environment. Thus user management is an important concept
- Even if the system is only used by a single user it is still important to create a user account besides the administrative (root) account
- *root* has unlimited privileges, many of which are not required for day to day activities
- Groups allow the grouping of individual users under a single name for file access control

Password and Group Files

- */etc/passwd* stores user account information
- */etc/group* stores group and membership information
- */etc/shadow* shadows the password file and stores encrypted passwords and password expiry information
- Password file contains the following entries (one line per user):
  - User ID: system assigned number
  - Group ID: ID of the user's default group
  - Comment: a descriptive string, usually user's name
  - Home directory: full path to user's home directory
  - Default shell

Adding a New User

- Command: **useradd** [options] user
- Common options:
  - -c *comment*: comment stored in password file, usually user's name
  - -d *directory*: home directory name
  - -s *shell*: shell for the account
  - -g *initial_group*: user's initial login group
- Examples:
  - useradd joe (add user joe with default values)
  - useradd -s /bin/bash -c 'Joe Smith' joe (add user joe with supplied values)

Deleting a User

- Command: **userdel** [options] user
- Common options:
  - -r: deletes files in the user's home directory
- Example:
  - userdel joe (delete joe, preserving his home directory)

Adding and Deleting Groups

- To add a new group:
  - **groupadd** group
- To delete an existing group:
  - **groupdel** group
  - Users must be removed from a primary group before that group can be deleted

Note:

Use desktop specific GUI program for user management

Changing User Passwords

- Command:
  - **passwd** user
- Examples:
  - passwd (changes password for current user)
  - passwd joe (changes password for user joe)

File System Hierarchy Overview (I)

- The directory tree was designed to be breakable into smaller parts, each capable of being on its own disk or partition
  - ease of system administration such as backups and quotas
  - works well in a networked environment where machines share file systems
- The major parts are *root (/)*, */usr*, */var* and */home*
- Root directory (/) contains files for
  - Booting the system and bringing it to a state where other file systems can be mounted
  - File system repair tools

File System Hierarchy Overview (II)

- */usr* contains commands, programs, libraries, man pages and other unchanging files needed for operation
  - Files should not be machine specific – this allows the file system to be shared across a network
- */var* contains changing (variable) system files, including spool directories (print, mail, etc.), logs and temporary files
- */home* contains users' home directories
  - Separating these makes backups easier
  - A large */home* may be separated further, e.g. */home/students* and */home/staff*

File System Hierarchy Overview (III)

- */etc* contains system configuration files
- */dev* contains device files
- */proc* is a special (virtual) file system created in memory to provide information about the system

File System Types

- Different file system types include:
  - ext3 – the default Linux file system (journalling file system)
  - ext2 – the file system used by older Linux versions
  - iso9660 – the standard cdrom file system
  - vfat / fat32 – Used by Windows95/98/XP
  - NTFS – used by Windows NT/XP
  - smbfs – SMB (Windows-compatible) system for shared drives
- Linux supports many file system types including those in the list above. Linux does not currently support writing to NTFS filesystems, so NTFS file systems are read-only

Using Storage Devices

- Storage devices are referred to by files in the */dev* directory. These files are categorised for easy naming
- *hd* devices refer to hard drives. These are suffixed by a character identifying the hard drive and a number identifying the partition on that hard drive. eg. The first partition on the third hard drive would be *hdc1*
- Other common prefixes are *fd* for floppy disks and *sd* for scsi and usb devices
- In order for Linux to access a storage device, its file system type must be specified, and it must be linked into the current directory hierarchy. This process is known as *mounting* a device

Mount Points

- Since Linux does not use the concept of *drives*, the file system consists of a single hierarchy, stemming from the root directory
- Additional file systems are mounted onto an existing directory, creating the illusion of a single file system
- The directory in the original file system that the new file system is mounted on is called the *mount point*

Mounting Devices (I)

- The *mount* command is used to mount and unmount file systems
- *mount* accepts as parameters the device to be mounted and the directory to which it must be linked – the mount point
- The file system type is defined using the *-t <filesystem>* option
- The format used is

mount -t <file system type> <device> <mount point>

Mounting Devices (II)

- Examples:
  - In order to mount the first partition on the first hard drive with an ext2 file system onto directory /drive2 we would type

    mount -t ext2 /dev/hda1 /drive2

  - To mount a USB memory stick:

    mount /dev/sda1 /mnt/flash

Determining Disk and Memory Usage

- The **df** command is used to determine how much free space is available on the mounted storage devices
- The **du** command shows how much storage space is being used by the current directory and all its subdirectories
- Common options for both:
  - -h: prints in human-readable format
- The **free** command displays usage information about physical memory and swap space

Locating files

- Command: **find** path -name pattern
- Examples:
  - find . -name "*.txt"
    (find *.txt* files starting from the current directory)
  - find / -name "*.rpm"
    (find rpm files starting from the root directory)
- Command: **locate** pattern [uses the (s)locate database, which needs to be updated regularly]
- Example:
  - locate txt (find any file whose name contains the string “txt”)

Querying File Types

- Command: **file** [options] file
- Common options:
  - -z: filters the file through gzip
- Examples:
  - file main.c
  - file index.html
  - file somearchive.tar.gz

Session 6 Command Summary

Command	Description
`useradd`	create a new user account
`userdel`	delete a user account
`groupadd`	create a new group
`groupdel`	delete a group
`mount`	mount a file system
`df`	summarise file system disk space usage
`du`	calculate file disk space usage
`free`	display information about free and used memory
`find`	search for files in the file system
`locate`	query the locate database for files
`file`	determine a file's type

Session 7 Overview

- Networking basics
- Configuring network devices
- Routing basics
- Host name resolution
- Startup sequence
- Service scripts

Networking Basics

- Each machine on a network is assigned
  - A host name, made up of a machine name and a domain name e.g. neptune.cs.wits.ac.za
  - An IP address. In the case of a server the IP address must be public and unique e.g. neptune.cs.wits.ac.za's IP address is 146.141.27.226
  - A network address, which specifies which other IP addresses form part of the same network
- An IP address is assigned to a physical interface such as an ethernet port

Host Names

- Host names provide a means to address a specific machine
  - This is necessary to locate dedicated services, e.g. web sites, ftp servers (www.google.com; ftp.is.co.za)
  - Host names are easier to remember than IP addresses and allow IP addresses of hosts to be easily changed
- Host names are resolved into IP addresses through
  - Domain Name System (DNS): a distributed registry of host name to IP address mappings and reverse mappings
  - Local */etc/hosts* file

IP Addresses

- Every machine on a network must be assigned an IP address
- IP addresses can be
  - *static*: fixed to a particular machine
  - *dynamic:* belong to a pool and bound to a machine at boot time (current implementation called DHCP – Dynamic Host Configuration Protocol)
- Servers have static IP addresses
- Clients (workstations) may have either – dynamic addresses are arguably easier to administer

Configuring Network Interfaces (I)

- Command: **ifconfig** *interface* [parameters]
- Frequently used parameters:
  - address: the interface's IP address
  - **netmask** *mask*: the associated subnet mask
  - **up**: actives the interface (implied if address is given)
  - **down**: deactivates the interface
- Used without parameters, the current configuration is displayed

Configuring Network Interfaces (II)

- Examples:
  - ifconfig eth0
    displays configuration for default ethernet card
  - ifconfig eth0 146.141.27.155
    sets the IP address and enables the interface
  - ifconfig eth0 146.141.27.155 netmask 255.255.255.0
    sets the IP address and the network mask
  - ifconfig eth0 down
    disables the ethernet interface

Routing (I)

- Routers use routing tables to route network traffic from one network to another (and throughout the Internet)
- Routers may be dedicated equipment, but Linux servers can also be set up as routers – this is beyond the scope of this course
- All networked machines need to be configured to determine where to send network traffic not destined for the local network – this is done by configuring a default route / gateway

Routing (II)

- Command: **route** [**add** | **del**] options
- **route** with no options displays the routing table
- **route add** adds a new route to the routing table
- To configure a default route, use the following command:
  route add default gw *<IP address>*
- For example, route add default gw 146.141.27.1

Host Name Resolution (I)

- Most machines are configured to resolve host names through the DNS
- For hosts that are not in the DNS (such as small networks with no DNS server) a local file (*/etc/hosts*) can be used to store host information as well
- The file */etc/host.conf* configures the order in which these 2 methods are applied to resolve host names. The standard configuration is `order hosts, bind` which first looks at the local file before querying the DNS
- BIND (Berkeley Internet Name Domain) is the most common name server implementation

Host Name Resolution (II)

- Information about name servers in the DNS to be queried is specified in */etc/resolv.conf*
- A sample file is

```
search cs.wits.ac.za
search ms.wits.ac.za
nameserver 146.141.27.9 dns
nameserver 146.141.15.210 caesar.wits.ac.za
```

- At least one name server should be specified
- The *search* option allow short names relative to the domain name to be used


Host Name Resolution (III)

- The **dig** and **nslookup** commands are used to query name servers
- For example

  ```
  nslookup neptune.cs.wits.ac.za
  ```

  produces

  ```
  Name: neptune.cs.wits.ac.za
  Address: 146.141.27.226
  ```

- Both commands have a variety of different options – consult the man pages for information

Network Troubleshooting

- The **ping** command sends ICMP echo request packets to the specified host and reports on how long it takes to receive a corresponding ICMP echo reply, e.g. ping neptune.cs.wits.ac.za
- The **traceroute** command attempts to display the route over which packets must travel to reach the destination
- Both commands do not work as effectively as they once did since firewalls nowadays often block out ICMP traffic (to prevent denial of service attacks)
- The ping command is useful for testing whether a newly connected machine can see others on the same network (e.g. by pinging the default gateway)

Startup Sequence

- The first program that runs when the computer boots is responsible for loading the operating system and is known as the bootloader
- Most Linux systems currently use the *grub* bootloader. *lilo* (linux loader) was its predecessor
- *grub* loads the *kernel* of the Linux operating system. It can be configured by editing the */etc/grub.conf* file
- The kernel then starts the *init* program which is responsible for starting all services and initial programs

Init and Runlevels

- The *init* process executes all the scripts that should run when Linux starts. The list of programs that should be run is customisable
- The *init* configuration is stored in */etc/inittab*
- */etc/inittab* file defines different modes (called runlevels) that the operating system can run in
- Associated with each runlevel is a set of programs which *init* should run at startup
- The default runlevel is set by the system administrator (and can be changed by editing the *initdefault* line) in */etc/inittab*

Runlevels

- Possible runlevels are:
  - 0: system halt (do not set *initdefault* to this)
  - 1: single-user mode
  - 2: multi-user mode, without remote network (incl. NFS)
  - 3: full multi-user mode
  - 4: unused
  - 5: full multi-user mode with network and X display manager
  - 6: system reboot (do not set *initdefault* to this)

Startup Scripts

- Startup scripts are located in the */etc/init.d/* directory (for Suse and Ubuntu – this differs from one distribution to another)
- Symbolic links in directories corresponding to the runlevel indicate which services should be started at each runlevel
  - /etc/init.d/rc3.d/ for runlevel 3
  - /etc/init.d/rc5.d/ for runlevel 5
- Links prefixed by S are run at startup (in increasing order)
- Links prefixed by K are run at shutdown (in decreasing order)

Starting and Stopping Services

- Linux services can be started and stopped manually by running the corresponding script with the arguments *start* or *stop*. e.g:
  - /etc/init.d/httpd stop
  - /etc/init.d/network start
- Startup scripts also optionally support the following options:
  - *restart*: stops (if running) then starts the service
  - *reload*: reloads the configuration without restarting the service
  - *force-reload*: reloads configuration if possible, otherwise restarts
  - *status*: shows current status of service
- Information about service processes is also always available through the **ps** command

Service-Related Commands (I)

- *chkconfig* is a convenient method of modifying the services automatically started up at each runlevel. It changes the symbolic links in */etc/init.d/rc*.d* according to the specified configuration. It supports the following options :
  - *--list* **:** lists known services and their current configurations
  - *--add <name>*: adds a service for configuration
  - *--del <name>*: removes a service
  - *--level <number> <name> <on/off/reset>*: configures a particular service on a specific runlevel. Services can be enabled or disabled at a particular runlevel using *on* or *off*. *reset* changes the configuration of the service to that specified in its initial configuration file

Service-Related Commands (II)

- *netstat* provides a variety of network-related information
- When run with no options, *netstat* displays all open sockets, i.e. shows all active connections on the machine, including local connections between processes
- Common options include:
  - *--tcp* : displays only tcp sockets
  - *--udp* : displays only udp sockets
  - *-l* : displays only listening sockets
  - *-r* : prints out the routing table
  - *-p* : shows the programs currently using particular sockets

Session 7 Command Summary

Command	Description
`ifconfig`	configure and display network interfaces
`route`	show and configure the routing table
`dig`	DNS lookup utility
`nslookup`	interactive DNS query tool
`ping`	send ICMP echo requests to network hosts
`chkconfig`	update and query runlevel information for services
`netstat`	report network connections, routing tables, etc.

Session 8 Overview

- Software management
  - Packaging and dependencies
  - Common package formats
- Compiling from source
- Managing software with RPMS
- Linux distributions
- Acquiring Linux and open source software
- Support and documentation

Why Software Management?

- Software installation and upgrades from the current distribution
  - Installing previously uninstalled software
  - New versions of software continuously released
  - Distribution upgrades
- New software – Linux distributions are bundled with a large amount of software, but
  - not all software can be distributed due to the vast amount of available software
  - they do not contain proprietary software, which you may acquire and need to install
  - do not generally contain niche application software

Packaging Software – Tarballs

- Software must be packaged in a convenient way to distribute or download
- The oldest and most generic format is the tarball (.tar.gz or .tar.bz2)
  - a tarred, compressed archive containing the program source or binaries (binaries are limited to a specific platform)
  - source tarballs are distribution (and sometimes platform) independent
  - but, usually the hardest to install (due to dependency issues and non-standard infrastructure)
- Niche software is unfortunately often only available in source tarballs

Packaging Software – Packages

- Packages are a distribution-specific method for distributing software
  - Are associated with a software (package) management system
  - Can have embedded pre- and post- installation scripts
  - Usually associated with binary installations (no need to compile)
  - RedHat package format (RPM) is the most widely supported
- Package managers
  - Manage software dependencies between packages
  - Simplify software management (installing, upgrading, removing)
  - Are tied to a specific distribution of Linux (unfortunately)

Software Dependencies (I)

- Scenario 1
  - You install a custom package that installs with additional shared software which was not obtained from your distributor, which the custom software is built against
  - The distribution's versions of the shared software breaks when the new software version gets installed and the distributor's version get uninstalled
- Scenario 2
  - You install a custom package which relies on shared software
  - You then install software from the distribution which has a different version of the shared software as a dependency
  - Your custom package breaks without your knowing why
  - If you reinstall the custom package, it overwrites the shared software from the distribution and a vicious cycle occurs

Software Dependencies (II)

- The moral of the story:
  - Always try to obtain software provided by the distribution
  - If this is not possible, try to obtain the software in the package format supported by the specific release of your distribution. (Another option – expert option – is to get the source package and create the package yourself)
  - If the only option is to compile from tarballs, either
    - Install into your own *~/bin* directory and add this directory to your path, or
    - Install into */usr/local/* (not into */usr*)

Compiling from Source (I)

- Look out for the following files at the top of the source code tree:
- *README*
  - should always be read first
  - contains information about software functionality, supported operating systems, dependencies on other software, installation instructions, authors and license of the software
- *INSTALL*
  - information about how to install the software
  - may contain information for different installation and architecture types

Compiling From Source (II)

- *TODO*
  - information about functionality to be added in the future
- *configure*
  - script that checks the configuration and settings of the machine
  - creates a Makefile used to compile the software
  - incredibly useful but not always available
- *Makefile*
  - specifies the procedure for compiling the software
  - quite technical but commonly used software does not require user interaction

Compiling From Source (III)

- Vanilla installation procedure looks as follows:

```
./configure
make
sudo make install
```

Common Package Formats

- RPMs
  - Supported by many distributions and probably the most common package type
  - Note that distributions often package their own RPMs so RPMs are not necessarily compatible across RPM-supporting distributions
- .DEBs
  - Debian-style package management with a versatile set of software management and reporting tools (text and graphical)

RPM Package Names

- Package names have strict naming rules, which contain the following information from left to right:
  - Name: package name
  - Version & Release number
  - Architecture: Intel architecture is i386
  - .rpm extension
- Examples:
  - gzip-1.3.3-9.i386.rpm
  - mozilla-1.2.1-26.i386.rpm
- *rpm* command is used to install, remove, upgrade, query and verify packages

Installing and Upgrading RPMs

- Command:
  - **rpm** -i packagefile
  - **rpm** -U packagefile
- Common options:
  - -h: uses hash marks to indicate progress
  - --test: verifies the installation without installing
  - -v: sets verbose mode
  - --nodeps: skips dependency checking (not recommended)
- Examples:
  - rpm -i mozilla-1.0.1-24.i386.rpm
  - rpm -Uvh gzip-1.3.3-5.i386.rpm

Uninstalling RPMs

- Command:
  - **rpm** -e package
- Common options:
  - --nodeps: skips dependency checking (not recommended)
  - --test: verifies the uninstall without uninstalling
- Example:
  - rpm -e mozilla-1.0.1-24

Querying Packages (I)

- Command: **rpm** -q
- Common options:
  - -a: displays a list of all packages installed
  - -f file: displays which package contains the specified file
  - -i package: displays information about an installed package
  - -c package: lists configuration files in an installed package
  - -d package: lists documentation files in an installed package
  - -l package: lists all files in an installed package
  - -R package: lists packages on which this package depends
  - -p packagefile: used in conjunction with other options, refers to (uninstalled) package file rather than installed package

Querying Packages (II)

- Examples:
  - rpm -qa (generates a list of all packages installed)
  - rpm -qi mozilla-1.0.1-24 (displays information about the installed mozilla package)
  - rpm -qpi mozilla-1.0.1-24.i386.rpm (displays information about the uninstalled package file mozilla-1.0.1-24.i386.rpm)
  - rpm -ql mozilla-1.0.1-24 (lists all files in the installed mozilla package)

YaST

- YaST (Yet another Setup Tool) is Suse's system and software configuration management tool (a front-end for configuring just about everything in the system)
- YaST's software manager is a front-end to the underlying RPM framework
  - Manages multiple dependencies concurrently
  - Allows for online updates from official Suse sources
  - Keeps track of installed and available software from CD and online sources
  - Provides a convenient mechanism for keeping uptodate with security patches and software updates

Acquiring Open Source Software (I)

- The safest place to acquire new software is from the distributor of your distribution (also remember that software you require may be on the original CDs)
- Sourceforge (sourceforge.net) is the largest repository of open source projects, but requires critical evaluation
- Open source indexes and search engines include
  - Freshmeat – www.freshmeat.net
  - Tuxfinder – www.tuxfinder.com
  - RPM search engine – www.rpmfind.net
- Bioinformatics.org (www.bioinformatics.org) is a repository for bioinformatics-specific software

Acquiring Open Source Software (II)

- Some project specific sites:
  - Apache Web server: www.apache.org
  - OpenOffice office suite: www.openoffice.org
  - PostgreSQL database: www.postgresql.org
  - MySQL database: www.mysql.com
  - GNU project: www.gnu.org
  - Mozilla Web browser suite: www.mozilla.org
  - GNOME desktop project: www.gnome.org
  - KDE desktop project: www.kde.org

Linux Distributions

- Many disparate efforts to package software needed for a complete Linux system has resulted in many different distributions
  - Caldera OpenLinux: http://www.calderasystems.com/
  - Debian GNU/Linux: http://www.debian.org/
  - Impi: http://www.impi.org.za/
  - Knoppix: http://www.knoppix.net/
  - Mandrake: http://www.linux-mandrake.com/
  - RedHat / Fedora: http://www.redhat.com/ & http://fedora.redhat.com/
  - Slackware: http://www.slackware.com/
  - Suse: http://www.suse.com/
  - Ubuntu: http://www.ubuntulinux.org/

Acquiring Linux

- Open Source Linux distributions are available from a number of different sources:
  - Almost always available on the Internet (and may have local mirrors)
  - Available through local distributors
  - From a friend with a CD burner...
  - Through libraries, community centres etc.
- Note that some “enterprise” versions contain proprietary software

Open Source Software Support

- There is a misconception of a lack of open source and Linux support
- In fact there are two routes for support: standard, paid-for support and the traditional community support
- Community support can be found through online documentation, mailing lists, discussion forums, IRC channels, user groups
- Linux documentation is also improving
  - Ad-hoc documentation on the Web
  - Distribution-specific manuals and online documentation
  - Books (stores and online – O'Reilly publishes many for free)

Selected Online Resources

- *www.linux.org*: general source of information pertaining to Linux
- *www.tldp.org*: (The Linux Documentation Project) official repository of technical documentation
- *www.slashdot.org*: popular news and discussion forum site
- *www.tectonic.co.za*: local news site featuring latest open source developments
- Distribution-specific sites: e.g. *portal.suse.com* provides Suse documentation
- *www.google.com* as always...