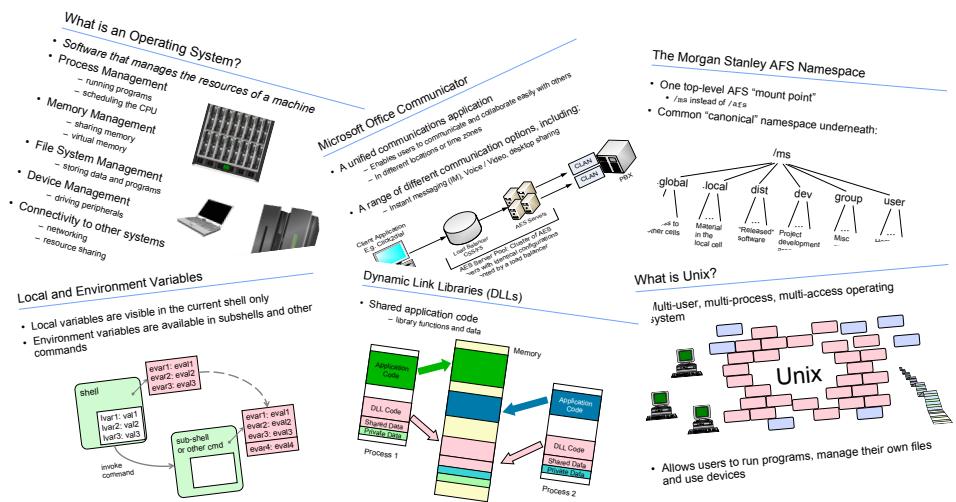


Operating Systems

Technology Analyst Program
August 2016



Operating Systems

Authors: George Ball, Michael W. Branco, David Mallon, Ewan Smith,
Paul Storer-Martin

Editor: David Mallon

Credits: George Ball, Barry Levine, David Mallon, Ewan Smith

Copyright © 2016 in parts:

Mallon Associates International Limited
Morgan Stanley, Inc.

All rights reserved. No part of this work covered by copyright hereon may be reproduced in any form or by any means; graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owners.

All products or services mentioned in this document are identified by the trademarks or service marks of their associated companies and are respectfully acknowledged.

Whilst every effort has been made to ensure the accuracy, adequacy and completeness of the information and material contained in this document, the document is provided "as is", without warranty of any kind, express or implied.

53 47 22 8 7 116 1 220 1

This booklet, the full-text book and the interactive web documents were built with Mallon Associates Opus.

Contents

- 1 | OS Fundamentals 4
- 2 | Unix/Linux Concepts 28
- 3 | Working with Unix/Linux 42
- 4 | The File System 58
- 5 | Network File Systems 76
- 6 | Working with Files 89
- 7 | File Access Control 111
- 8 | Unix Editors 128
- 9 | Unix/Linux Shell 149
- 10 | Shell Variables 167
- 11 | Controlling Processes 192
- 12 | Utilities and Filters 208
- 13 | Windows Basic Architecture 228
- 14 | Desktop & Tools 248
- 15 | Application Infrastructure 270
- 16 | End User Computing 282
- 17 | Introduction to Mainframes 298

1 OS Fundamentals

- What is an Operating System? 5
- What is a Process? 6
- Process States 7
- Preemptive Multitasking 8
- Preemptive Multiprocessing 9
- Multiple Processors 10
- Multithreading 11
- Memory Management 12
- Virtual and Physical Memory 13
- File System Management 18
- Example: Fetch Data from a File 20
- Device Drivers 24
- Networking 25
- Distributed Services 26
- Clustering 27

What is an Operating System?

- *Software that manages the resources of a machine*
- Process Management
 - running programs
 - scheduling the CPU
- Memory Management
 - sharing memory
 - virtual memory
- File System Management
 - storing data and programs
- Device Management
 - driving peripherals
- Connectivity to other systems
 - networking
 - resource sharing



The operating system is a critical part of any computer system, although end users are seldom aware that it is even there.

Its job is essentially one of resource management, providing an interface between the applications that are running on a system and the hardware that they are using. The operating system must ensure that each application gets a "fair" share of the resources, while at the same time ensuring that one application does not corrupt any work being done by another.

The main resources managed by an OS are

The CPU (or CPUs in large systems), which provide the basic execution resource for each application. Sharing the CPU amongst different tasks that are running at the same time is known as multitasking.

The memory of the system. If there are many applications running at the same time, each will require memory to store variables, etc. The physical memory in the system must be partitioned and shared amongst the tasks.

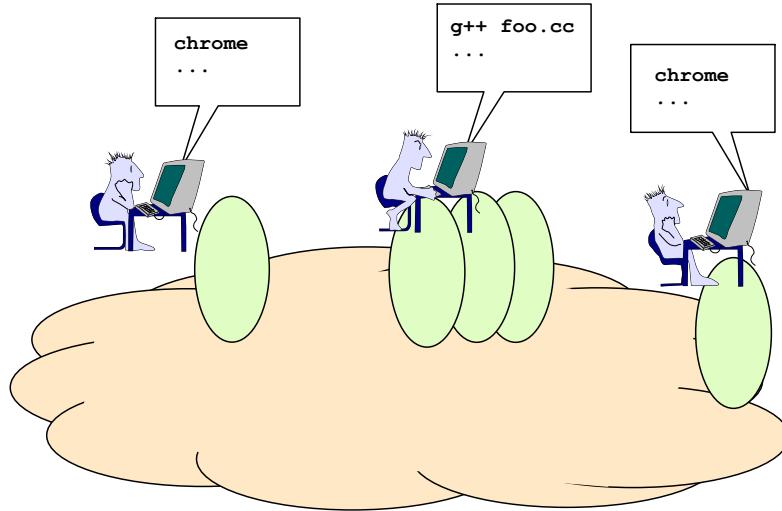
The file storage of the system, giving long term storage capabilities to applications.

The physical devices attached to the system, for example printers, scanners and modems as well as internal devices such as disks. Often programs need direct access to these devices.

Network capabilities, allowing systems to communicate with each other. This could be viewed as a special case of device accessing as described above, but networking and distribution is such an important part of today's systems architectures that it is correct to treat it separately.

Different operating systems have different ways of achieving the above resource management, but they nevertheless all have to perform the same basic tasks.

What is a Process?



- A process is a program in execution
 - executes “user” instructions
 - requests operating system services when needed

One of the most important concepts in operating systems is the process. The traditional model of a process is that of a program in execution. A program is essentially a static object - a file stored in the system containing instructions for execution.

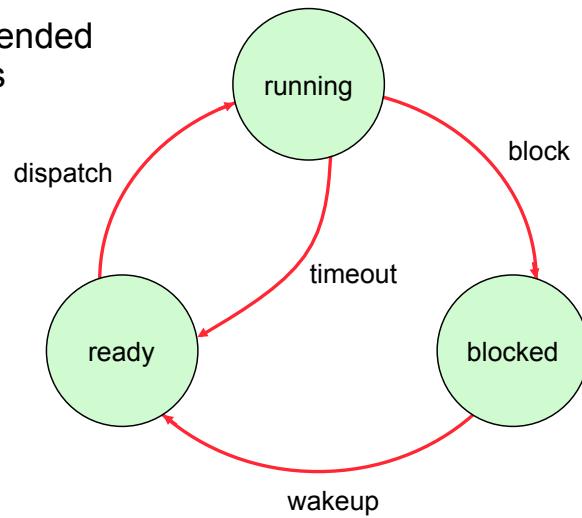
When the program is executed, the operating system has to set up and maintain resources for it, such as memory to hold its data, system resources such as file access handles and identification information so that access permissions can be verified. A process is a container for this information and for (at least part of) the code to be executed.

Most modern operating systems support multiple concurrent users, and by implication multiple concurrent processes. In such systems, the same program may be executed more than once at the same time, leading to several different processes all associated with the same program file. In the example, two of the users are executing the chrome command, resulting in two chrome processes.

Most of the code that is executed in a process is that of the actual program. This is known as "user code". The user code of each process is well insulated from that of other processes. However the process may need the operating system to do some work for it. The process makes a call into the operating system, causing operating system code to run on its behalf. When this happens the process is said to be running in kernel mode.

Process States

- Most operating systems support the simultaneous execution of multiple processes
- Processes are suspended on slow I/O requests
- Processes share the CPU by time-slicing



Operating systems support the idea of multiple processes running concurrently. Such systems are called multiprocessing, multitasking or time-sharing systems. It is not necessary for there to be multiple CPUs. However, when there are fewer CPUs than jobs (processes) to run, the work must be scheduled amongst them.

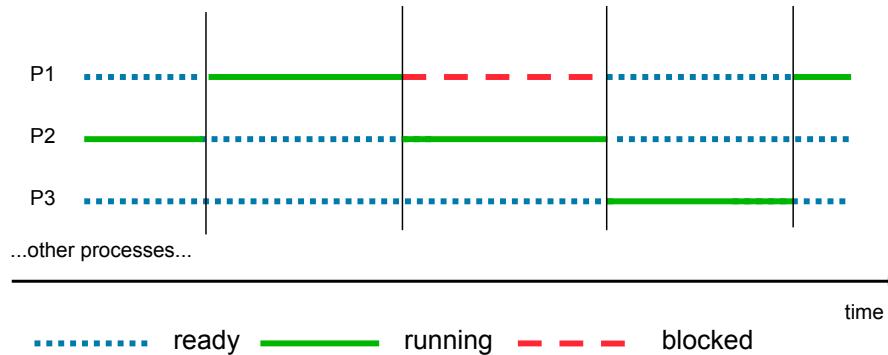
To give the illusion that all of the processes are running simultaneously, each process will usually run for a very short period of time, and then give up (or lose) the CPU to another process that is able to run. A process may also give up (or lose) the CPU if it performs a slow operation, such as communicating with an I/O device. Such processes are said to be blocked (sleeping or waiting) until the I/O request is satisfied. Processes ready and waiting to run are said to be ready (or runnable) and are maintained by the process manager in some form of queue.

The order in which processes are dispatched (given access to the CPU) from this queue depends on the scheduling algorithm being used, and may be controlled by process priorities. Some operating systems allow more than one scheduling policy to be active.

If the computer has multiple CPUs, or multiple cores per CPU, then we can have multiple processes running simultaneously, each following the state transition rules described above/

Preemptive Multitasking

- Operating system schedules processes
 - each running process is passive
 - operating system interrupts processes when time expired



- Assumes one CPU

With preemptive multitasking the operating system takes full control of process scheduling. Processes are given a time period (often called a quantum since it is very short) in which to operate, at the end of which they are suspended so that another waiting process may run.

A process may also lose control of the CPU if it performs a "slow" operation, like input or output.

When choosing which process to allow onto the CPU, the operating system will normally use an attribute of the process known as its priority, with a higher priority process being guaranteed access to the CPU before lower priority ones.

In the timing diagram above, process P2 is interrupted and control is passed to P1; this performs a slow operation and is blocked and P2 is once again given access to the CPU.

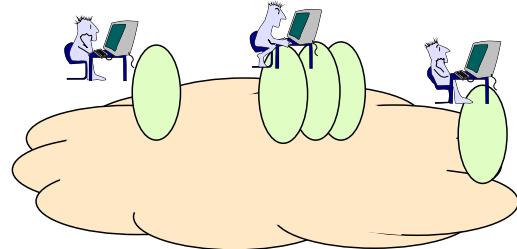
There are various schemes used to control how much CPU is allocated to each process. In the above picture, P2 runs at a higher priority than P3, which is why P2's time with the CPU is greater.

To avoid infinite starvation of the CPU, the process priority of P3 will normally increase while it is ready (and waiting) so that eventually it will be awarded time on the processor. P3's priority will then return to its former low value.

The diagram shown here assumes a single CPU. In modern, multi-core machines, we may have more than one process in the "running" state at a time.

Preemptive Multiprocessing

- Only one process can execute at a time
 - on single CPU
- Operating system chooses process
 - based on state...
 - ready ?
 - ...and priority
 - highest first
 - high priority process will preempt lower priority
- Process context determines what happens
 - i.e. where in the program the process executes
- One context per process
 - means one “thread” of execution can be managed



In a single CPU system, only one process can be in execution at a time. Remember that the kernel switches between processes very quickly to create the illusion that a lot is happening at the same time, but at any instant there will only ever be one process actually running.

The kernel must choose which process to run, based on the information that it keeps about the processes. In normal systems, the principle is very simple: at each context switch, the kernel always chooses the highest priority, runnable process to run. In real-time systems the decisions around scheduling may be different, reflecting the different response time requirements of the system.

To set a process running, the kernel loads the CPU registers with the process' saved context. This includes the address of the next instruction to execute from the program (Program Counter), and details of where the process' private memory can be found.

Essentially, the process context is a snapshot of exactly where in the process we are. It represents a view of the path of control through the process, often known as a thread of control.

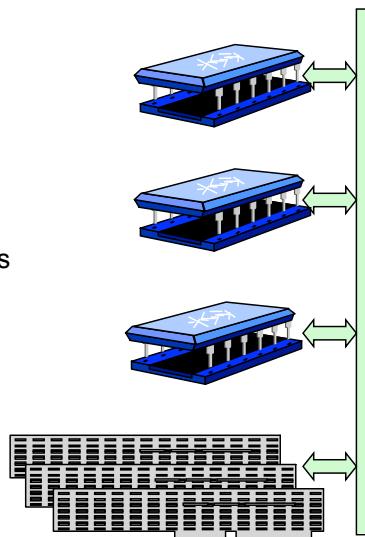
In traditional operating systems only one thread of control may be associated with a process. In other words, a process can only ever be doing one thing.

Multiple Processors

- Processes are scheduled across a family of processors
 - more than one process may execute at a time
 - increases system performance
 - system needs to be load balanced

- Symmetrical multiprocessing

- all processors are identical, no “master/slave” relationships
- communication through memory
- resilient to processor failure
- number of processors limited by memory contention



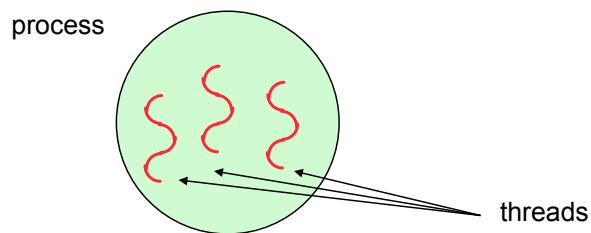
Multiple processor (MP) machines are able to run more than one process at the same time. They therefore offer the potential of increased system performance.

There are a number of architectures for MP machines, coupling or decoupling the CPUs from memory and each other. A form often used on low-mid range machines (PCs and Sun boxes) is symmetrical multiprocessing. Essentially, all the processors are the same and they share a common memory pool. The operating system schedules processes across the pool of processors; the shared memory is used to hold the shared program code and data.

A limitation of symmetrical multiprocessing is that the shared hardware resources, and especially memory, are points of contention. Symmetric multiprocessor systems usually have no more than 8 processors whereas a true multiprocessor parallel machine may have tens of thousands of processors (TMC connection machine).

Multithreading

- A process is sub-divided into multiple concurrent paths of execution
 - improves performance
 - a process can both wait and run at the same time



- Each is scheduled by the process manager
 - may be scheduled across multiple processors
 - process must manage shared memory resources

In traditional operating systems (early versions of Unix and most mainframe operating systems) a process is an indivisible unit of work with a single execution path.

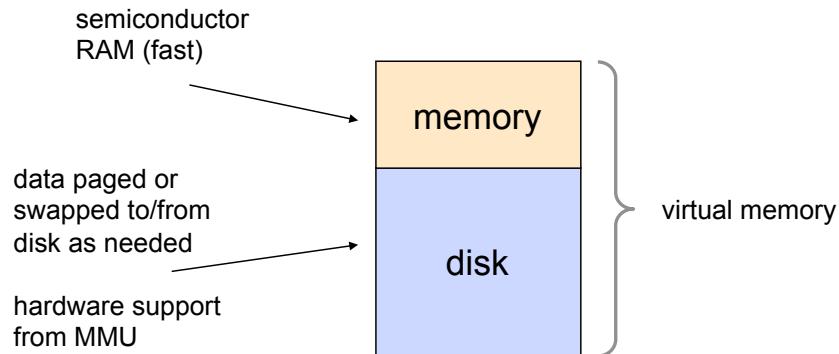
Modern operating systems support multithreading whereby a single process may have multiple concurrent threads of execution. For example, a multithreaded spreadsheet application may be able to recalculate a table, scroll the window and report on real-time data inputs simultaneously.

When run on multiprocessor systems, multithreaded processes may be able to execute on several CPUs simultaneously, since each thread has its own context. This will clearly increase the performance of the application. However, the amount by which the throughput increases will depend on how concurrent the program is, that is, how often different parts of the program must synchronise.

Multithreaded programs increase application performance even on single processor machines, since some threads in a program may be able to execute whilst others wait for slow events.

Memory Management

- Coordinates the allocation and sharing of memory
 - limited amount of RAM, large amount of disk
 - processes desire their own logical address space
 - processes must be protected from one another

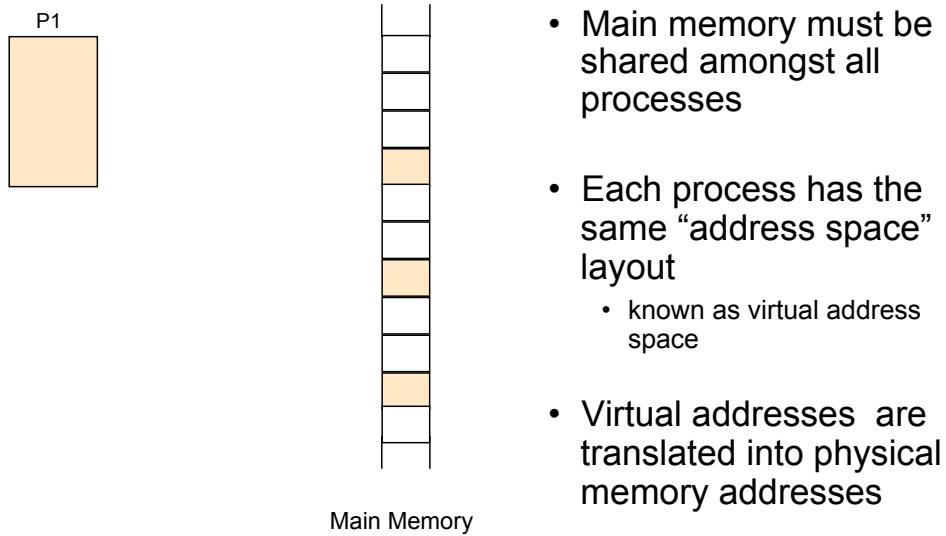


Another crucial job performed by the operating system is memory management. There is usually insufficient RAM for all of the processes running on a machine. Since RAM is relatively expensive, portions of disk (often called swap regions) are used to extend the amount of memory that is available to an application.

The total amount of memory that appears to be available for a process to use is called virtual memory. The memory manager is responsible for moving pages of processes into and out of memory as needed by the CPU. This will normally happen without the knowledge of the applications.

The memory manager gives each process its own view of memory, represented by a logical address space, arranges for this to be translated into physical memory resources and protects each process from accidentally overwriting other process' address spaces.

Virtual and Physical Memory



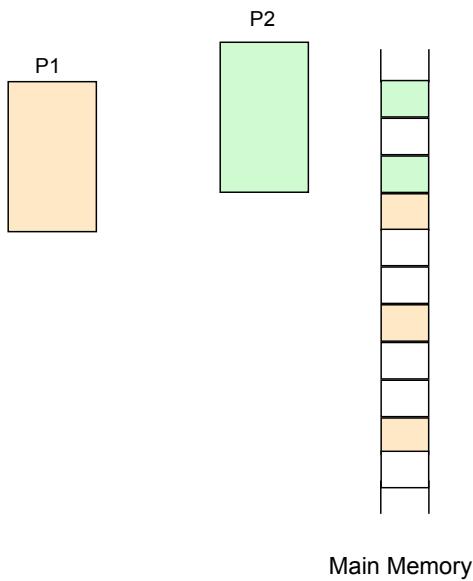
Within the operating system there will be many processes active at the same time. All processes have the same view of their address space.

The operating system and specialised hardware known as the memory management unit (MMU) must ensure that the physical memory resources of the computer are shared amongst all the processes that need it.

To do this, the addresses in the process address space, known as virtual addresses, are translated into real, physical memory addresses so that the data or instructions can be fetched from memory. To achieve this, physical memory is divided into small units known as pages. Process' virtual address spaces are also divided into pages, the same size as the "physical" pages. The system may then associate "virtual" or "logical" pages with physical pages.

The slide shows a potential scenario where a single process has been started, and has been allocated 3 pages of physical memory

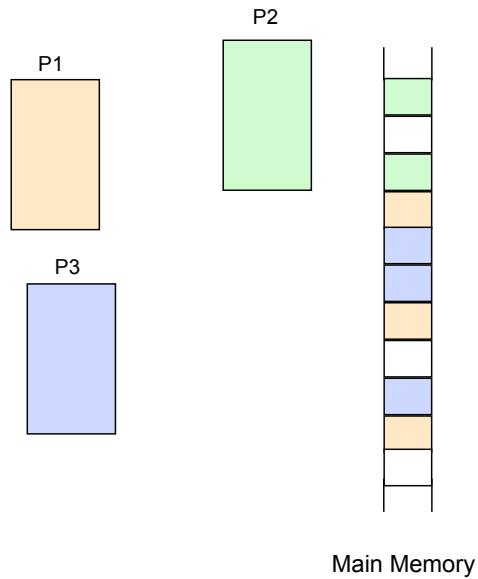
Virtual and Physical Memory



- Main memory must be shared amongst all processes
- Each process has the same “address space” layout
 - known as virtual address space
- Virtual addresses are translated into physical memory addresses

A second process, P2, is started and it is allocated two more pages from main memory.

Virtual and Physical Memory

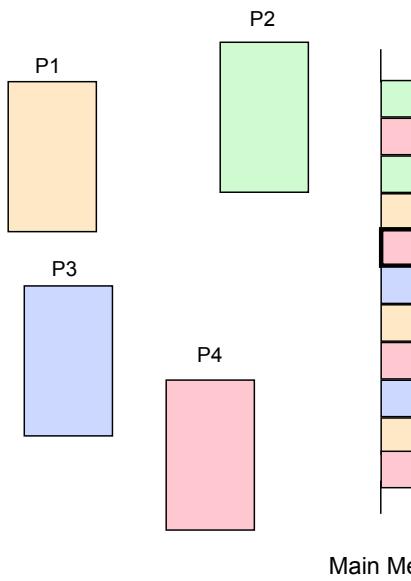


- Main memory must be shared amongst all processes
- Each process has the same “address space” layout
 - known as virtual address space
- Virtual addresses are translated into physical memory addresses

Now 3 processes are active, each has its own address space and each has certain parts of physical memory assigned to it.

Notice that the physical memory for a process does not have to be contiguous. There is no advantage in forcing the physical memory to be contiguous for a process, indeed it detracts from the system's ability to manage the memory effectively.

Virtual and Physical Memory



- Main memory must be shared amongst all processes
- Each process has the same “address space” layout
 - known as virtual address space
- Virtual addresses are translated into physical memory addresses

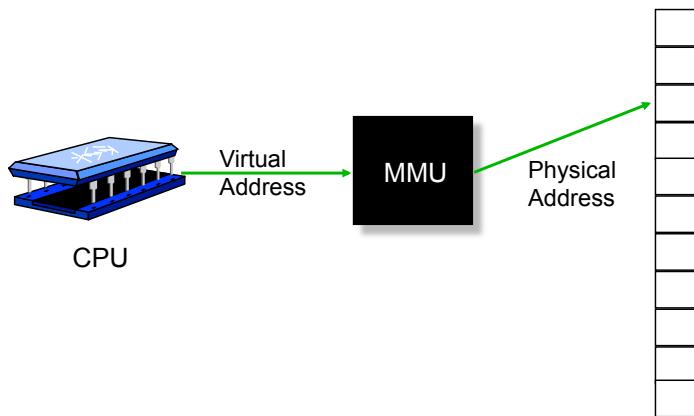
A fourth process is started, its memory requirements are such that there are not enough unallocated memory pages of physical memory, so the operating system "steals" a page from process P3 and allocates that page to P4.

The contents of the page will likely be copied out to the "swap" area of disk so that they can be restored in the future, should they be required again by P3.

Virtual and Physical Memory

- Role of the MMU

- different implementations of MMU on different architectures
 - Sparc
 - Intel



Whenever the contents of a location in memory are required, either to retrieve data or to retrieve an instruction, the CPU makes the request in terms of an address in the process' virtual address space.

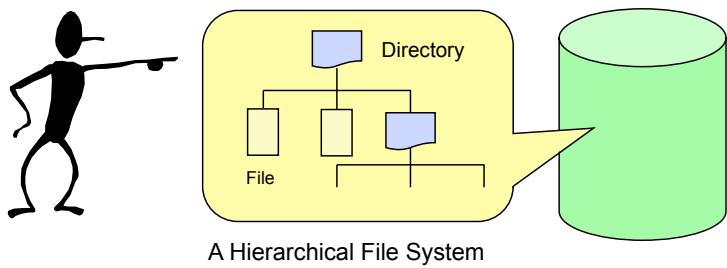
Specialised hardware known as the Memory Management Unit will translate this into a physical memory location, or indicate that the address is invalid.

If valid, the contents of that location can then be retrieved.

The MMU is implemented as some specialised hardware together with tables that keep track of how the translations are to be done.

File System Management

- Manages file system abstraction
 - the illusion that data and programs are stored in tidy files
- Modern operating systems often have file hierarchies
 - analogous to a filing cabinet, but with drawers within drawers
 - allows data and files to be easily organised and found



The file system manager is responsible for managing the long term storage facilities of the computer system. In most cases this involves accessing the disk(s) that are attached to the system.

In most operating systems, files are organised as an inverted tree structure, and users are able to navigate through the tree to the files they need. The internal nodes in this tree are known as directories.

Some systems allow multiple symbolic names to be created for a file or directory.

File System Management

- To understand what the file system manager does
 - look at some of the commands it supports in Windows and Unix

Unix	Windows	Action
<code>mkdir</code>	MD	<i>create a directory</i>
<code>ls</code>	DIR	<i>look at files in directory</i>
<code>cd</code>	CD	<i>move to another directory</i>
<code>cp</code>	COPY	<i>copy a file</i>
<code>cat</code>	TYPE	<i>examine the contents of a file</i>
<code>rm</code>	DEL	<i>delete a file</i>
<code>rmdir</code>	RD	<i>delete a directory</i>

- These commands act through the file system manager
 - it receives requests and works with the relevant storage device (via its device driver software) to complete the tasks

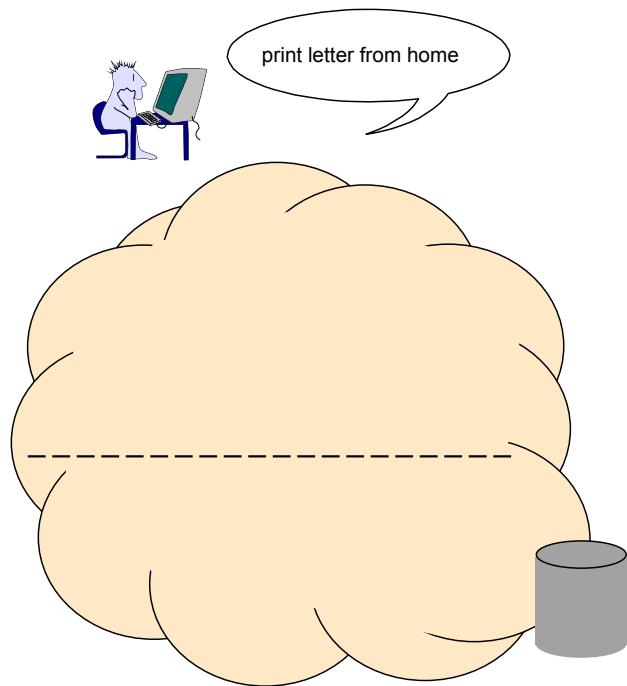
Users will normally be able to manipulate files and directories, although there will be access controls applied to make sure that one user is not able to corrupt another user's files, or any of the files necessary to keep the system running.

A file (or directory) will have a number of attributes that are used to implement these restrictions, for example an owner, and a set of permissions that govern what can be done to the file. It is usually also possible to discover the size of a file and when it was last accessed or modified.

The slide shows some of the common commands used in Unix and Windows NT/2000 to examine and possibly change the structure of the file system.

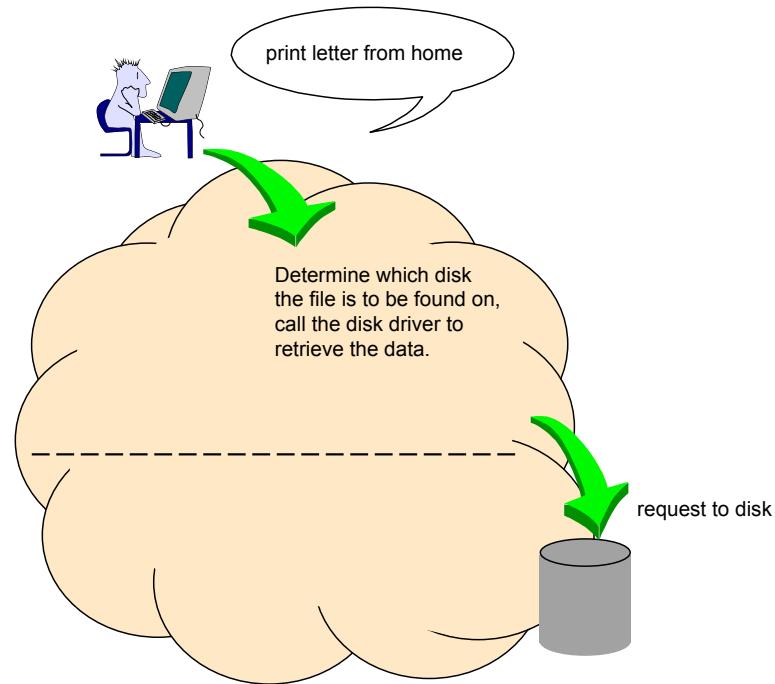
These days, most systems will also provide a graphical tool to help in this.

Example: Fetch Data from a File



In this simplified example a user wishes to read the content of a file.

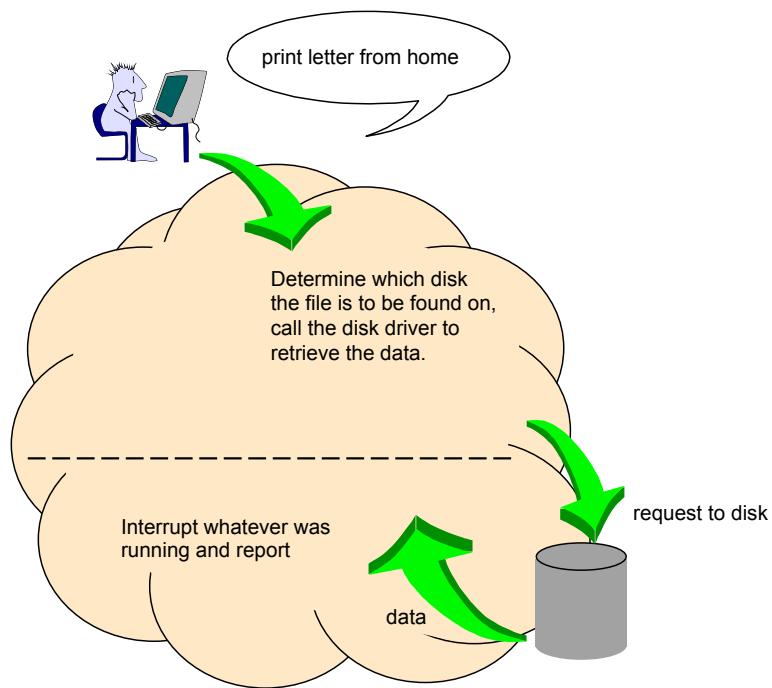
Example: Fetch Data from a File



The program being executed makes a call into the file system manager code within the operating system to ask for the data. This passes the request to the relevant storage device driver. Device drivers are modules of program code, inside the operating system, that know how to communicate with specific devices.

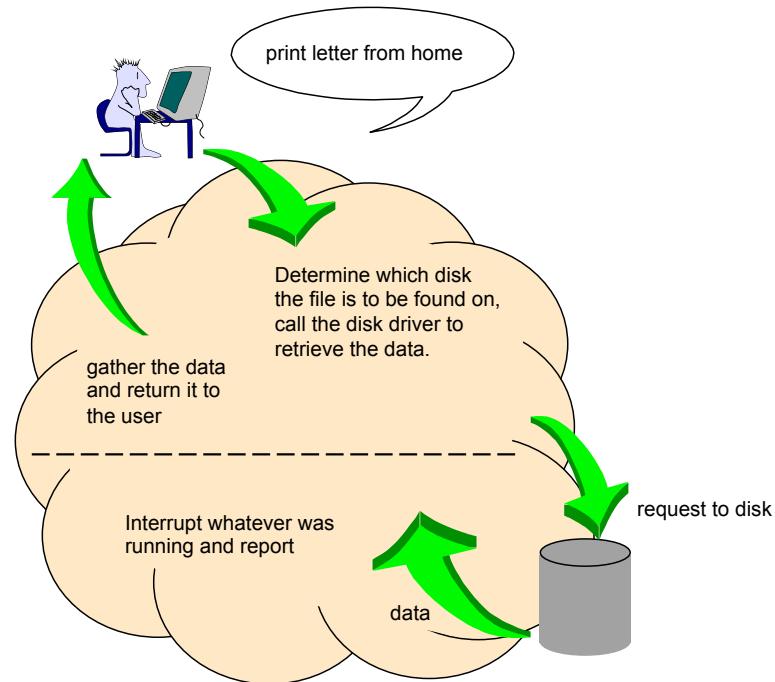
Once the request has been made, the calling process is blocked, which waits for the data to be returned.

Example: Fetch Data from a File



Once the data has been found, it is returned to the operating system. A hardware signal known as an interrupt is used to notify the operating system that the request has been completed..

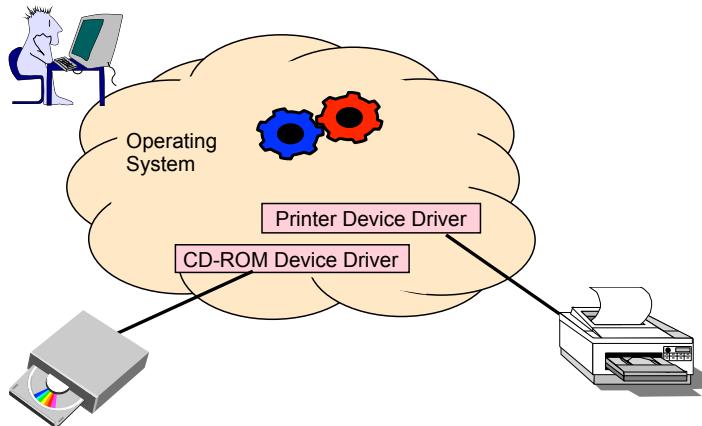
Example: Fetch Data from a File



The data is then gathered together for the process, at which point the process is able to continue.

Device Drivers

- Software modules to communicate with peripherals
 - helps operating systems send and receive data
 - provides a unifying abstraction for related devices



Applications will often require to communicate directly with devices that are attached to the computer. For example, an imaging application may require to read data from a scanner that is processing an input image such as a photograph. Alternatively a backup program would wish access to a tape drive or CD writer to copy the information that is being backed up.

Operating systems provide an abstract view of devices, that makes it easy to communicate with them. Otherwise, getting information onto or off a device can be extremely complex. This abstract view of devices is provided by a software module in the operating system called a device driver. Its job is to convert a common set of primitive operations into device specific functionality.

Before a computer running under the control of a specific operating system is able to access a device, there must be an appropriate device driver available. In most cases, device drivers for a given operating system are provided by the equipment vendor. However most operating systems publish APIs that allow developers to produce their own drivers for unsupported devices.

In some specialised cases, it is possible to have a device driver that does not communicate with a piece of hardware, but with another software module in the operating system, known as a pseudo-device.

Networking

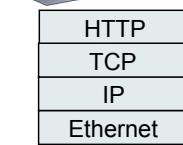
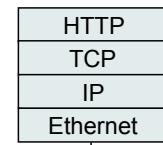
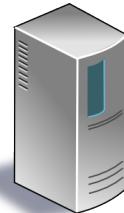
- Basic support for systems to communicate with other systems

- supports functionality provided by applications

- WWW

- FTP

- email



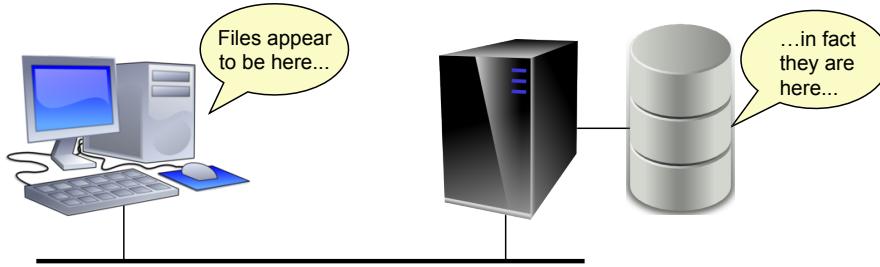
Nowadays, computers must be able to communicate with other computers. Many applications that run on systems rely on there being a reliable communications link to other systems.

A detailed discussion of networking topics is not possible here, however most of the communications between systems today is based on a set of protocols known collectively as the Internet protocols or TCP/IP. Application such as the World Wide Web, Electronic Mail and File Transfer all use the same core protocols, which are in fact implemented in the operating system.

Network protocol implementations can be viewed as specialised forms of device drivers (indeed they will ultimately involve communicating with a network interface device). However they are so important today that we can consider them as a component of the operating system in their own right.

Distributed Services

- Distributed systems allow resources sharing
 - printers
 - files
- Distributed file system
 - extension of file system ideas presented earlier
 - files may appear to be on one system, but are stored on another



Once we have basic connectivity between systems, it becomes possible to build much more sophisticated environments and applications. Relatively simple client/server applications such as email and the World Wide Web are only the start.

Modern operating systems will now implement resource sharing, so that many different systems can access the facilities on each other's computers. Two of the most obvious examples of resource sharing involve printers and files.

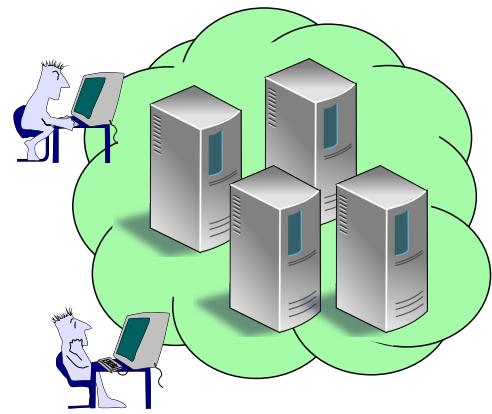
It is very easy to arrange for multiple systems to access a printer connected to a remote system as if it were connected locally.

It is also possible to transparently access files and directories on remote systems as though they were on the local disks of the system. To the "user" (end user or application) the file is accessed in exactly the same way whether or not it is on the user's local system. The illusion is created of there being more disk storage available than is actually the case.

There are many advantages to being able to centralise single copies of commonly accessed files in this way. There are also a number of problems that arise when accessing files remotely as opposed to locally, but these are outside the scope of this discussion.

Clustering

- Group multiple computers together so they appear as a single system
 - extends ideas of resource sharing to include CPU, memory and other devices
- Increases computing power available to applications
 - sometimes used as an inexpensive alternative to multi-processor hardware
 - usually “server” systems
- Greater fault tolerance
 - transparent “fail-over”



If we follow the ideas of resource sharing to their conclusion, we arrive at clusters of systems.

A cluster is a collection of systems that are operating together in such a way as to appear as a single system to users. The resources in terms of memory, CPU and so on are all combined together so that overall performance and throughput are substantially increased.

Clustering normally requires significant support within the operating system, although applications should be able to operate without knowing that they are in a clustered environment rather than a single system. The effect is similar to that of running the application on a system with multiple CPUs. However clustered systems are usually significantly cheaper to build than the corresponding MP systems since they usually do not require expensive hardware.

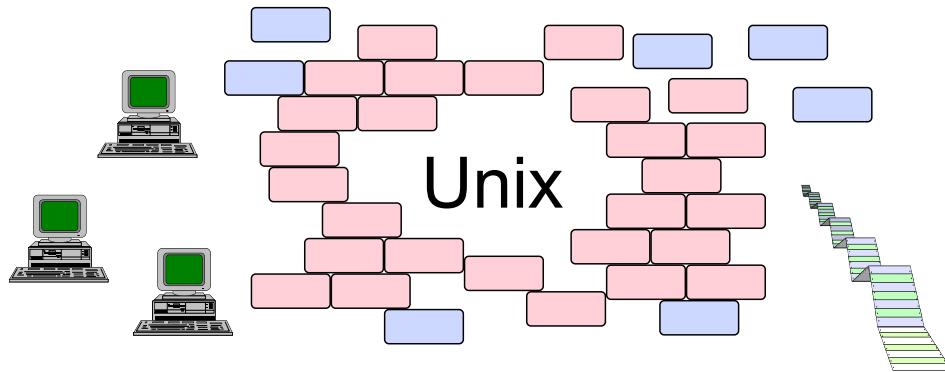
Clustered systems also offer increased fault tolerance, since if one system fails another can take over transparently.

2 Unix/Linux Concepts

- What is Unix? 29
- Major Features 30
- Unix and Linux 31
- Unix at Morgan Stanley 33
- Unix/Linux Key Concepts 34
- Files & Processes 35
- Organisation of Files 36
- Organisation of Processes 37
- Login 38
- What are Users? 39
- The Structure of Unix 40

What is Unix?

- Multi-user, multi-process, multi-access operating system



- Allows users to run programs, manage their own files and use devices

Unix is a multi-user, multi-process, multi-access operating system. This means that it can support multiple simultaneous users, each executing multiple programs.

Unix provides an operating environment for users to run programs, manage files, accessing devices, communicate with each other and co-ordinate their activities.

Unix is commonly used in networking environments, allowing data and resources to be shared amongst the connected machines.

Major Features

- Simple, powerful, user interface
- Complex commands are made from simple ones
- Hierarchical filesystem
- Consistent file format, the byte stream
- Simple, consistent, peripheral interface
- Hides machine architecture from user

Unix provides hundreds of commands each designed to do one thing well. Through a Unix shell (command line interpreter) collections of such commands are combined to perform complex tasks.

In Unix, files on disk, devices and the input and output of running programs are considered files. All physical devices have filenames, and behave as ordinary files.

The fundamental component of information in Unix is the byte stream. It allows files, devices and even programs to be used interchangeably as the source or destination of data; and thus allows the underlying machine architecture to be hidden from the user.

Unix and Linux

- Unix was originally distributed free of charge
 - mainly to university and research institutions
- Various companies formed around Unix
 - or adopted Unix as a "product"
 - commercialisation led to lack of "freedom"
 - BSD Unix still available as "free" version
- Free Software Foundation formed as reaction
 - aimed to develop and distribute "free" software for Unix-like systems
- Combination with work of Linus Torvalds produced Linux
 - variety of licenses support varying levels of "freedom"



The earliest versions of Unix were developed at Bell Laboratories, and were distributed free of charge to anyone who wanted them - including all of the source code. Most interest centred around universities and research organisations, for whom Unix provided a powerful platform both for teaching computer science and also for use in developing software to support other subject areas.

However it became clear that there was a potential commercial benefit in building and selling products that were based on Unix. AT&T, the "owners" of Unix commercialised it and a number of companies also formed to build and sell Unix related products. Perhaps the best known of these was Sun Microsystems which until its recent acquisition by Oracle, built and sold a powerful and successful range of Unix based computers. An offshoot of the original development stream was continued at the University of California, Berkeley, as BSD Unix, and was still distributed freely to universities. However by now there were many different systems, derived from Unix, that called themselves Unix, so that nowadays the term "Unix" is used to describe a whole family of operating systems that all share the key basic ideas of Unix from the early days.

Many people who used Unix objected to the commercialisation of Unix. One result of this was the formation of the Free Software Foundation, which aimed to develop and distribute a Unix-like system that was free of charge, and free of the commercial copyright restrictions of mainstream Unix. A combination of this software, which became known as GNU, with work done by Linus Torvalds on a freely distributable operating system kernel, led to a "product" that became known as Linux (more correctly GNU/Linux).

Unix and Linux

- Commercial and free versions are available

- Solaris

- Sun Microsystems, now Oracle

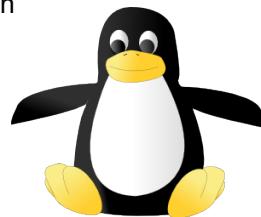


- Red Hat

- one of the first companies to "commercialise" Linux
 - Fedora is free version
 - Red Hat Enterprise Linux is commercial version

- Debian

- still strictly "open source"
 - variants such as Ubuntu, Mint very popular

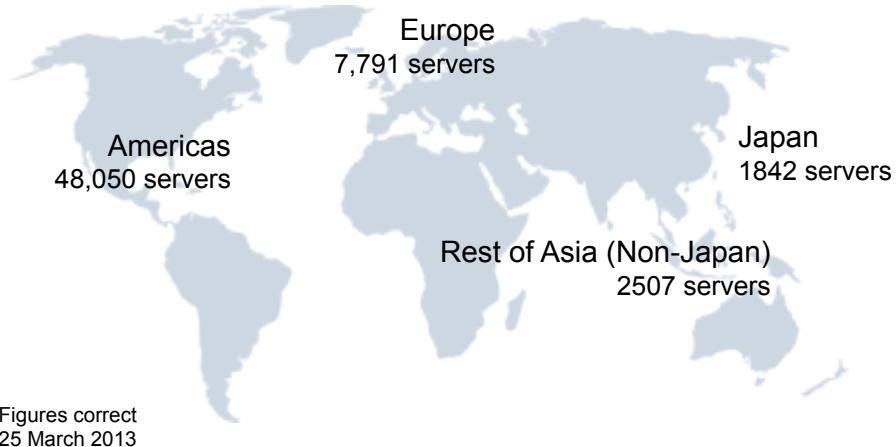


The success of Linux and its evolution into a powerful, stable system has led to a complete overhaul in the Unix marketplace in recent years.

Unix and Linux variants are often available in commercial versions, which are not free but are commercially tested and supported, with "free" or open source equivalents available for people to use as they see fit. Larger organisations will tend to use these products because of the commercial security they provide. The free versions are popular with small organisations, home users, and also as proving grounds for new features which can then be fed into the commercial distributions with a greater degree of stability.

Unix at Morgan Stanley

- Key strategic OS Platform
 - 60,190 servers (including grid compute plant)
 - 2% Solaris (Intel/SPARC), 98% Linux (Red Hat)



Unix has always been a key strategic operating system platform at Morgan Stanley.

Most Unix deployments are for server side operation, with Windows being the strategic desktop platform. By far the majority of systems in use are based on Red Hat Enterprise Linux, having moved from Solaris in years gone by. Indeed many Solaris systems are retained simply to support existing legacy systems that will not run on Linux.

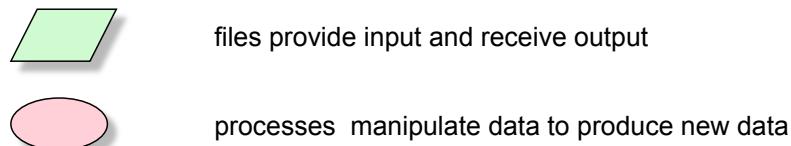
The move from Solaris to Linux was to a large extent driven by hardware costs, as Linux was available to run on commodity hardware such as Intel blade servers while Solaris at the time still required expensive proprietary hardware from Sun. The systems were similar enough that porting much of the existing application base was not difficult. Even after Solaris moved onto similar hardware platforms with Solaris 10 and OpenSolaris, its use has not really recovered inside Morgan Stanley.

Unix/Linux Key Concepts

- Computer programs process data



- There are only two entities in Unix



Unix/Linux owes its enduring success to its simple design ideas. The business of computer programs is to process data. To read data from some source, apply some computation, and generate result data. This model is supported directly by Unix.

In Unix there are only two entities: files and processes. Files represent the data being read or written, processes are the active entities reading, processing and writing data.

Files & Processes

- Everything is a process or a file
- Files are passive entities
 - streams of bytes stored on disks
 - interfaces to devices
 - the input and output streams of running programs
- Processes are active entities
 - instances of running programs
 - instructions for the CPU
- Every process starts life as a file
 - Unix commands are stored as program files on the disk

In Unix, everything is a process or a file. No other entities exist, not even disks, printers, terminals or networks---in Unix all of these things appear as files. The idea of using file names to represent devices saves introducing another concept. When a device's file is read or written, Unix ensures that the interactions are propagated to the particular device which the file represents.

Files are passive entities, unable in themselves to do anything. Processes are active entities, in some sense they have life. Consider a human as a process, and a suitcase as data. The suitcase cannot move itself, because it does not have life. A human must be applied to the suitcase in order for it to move. Likewise a process is applied to a file in order for it to be processed.

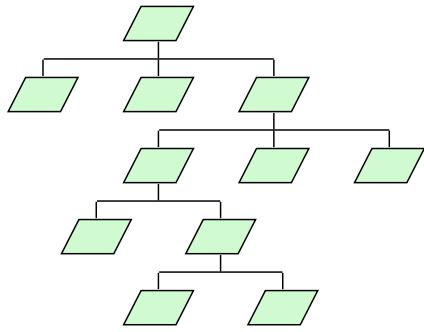
Unix is unusual compared with many operating systems in that process creation is relatively inexpensive. As a consequence, each command is executed as a single process. This differs from other systems which often run commands as procedure or function calls within a central command process.

In Unix, each invocation of a command gives rise to a new process. The process is created from the command's program file, executes and then dies. It is not unusual for one command to be executed by one or more users simultaneously, giving rise to many independent processes (instances) each performing the same task.

Note that the instructions as to what a process should do are stored in a corresponding program file on disk. When a command is issued, the text of its program file is used to make the process. The CPU interprets each instruction within the process to carry out the work.

Organisation of Files

- Files are organised as an inverted tree



The Unix File System consists of a single root directory which contains files.

Files may be data, programs devices or directories.

This logical file system may be composed of many physical devices and networks.

Both process and file entities in Unix are organised as trees. The tree used to hold files is called the Unix File System (UFS). The tree used to hold processes is simply called the process tree.

Unix File System

The Unix File System is organised as an inverted tree; the root is at the top, and branches and leaves in the form of directories and files grow down. A directory is a special file which can hold other files. Since these files may themselves be directories, a tree structure is formed.

Unix systems contain only one logical file system. The file system may span multiple partitions and disks, cross networks and exist in multiple physical forms. However, the illusion maintained by the operating system, is that the file system is one, coherent, tree. As a user moves around the file system, Unix ensures that the physical joins between disks or networks remain hidden.

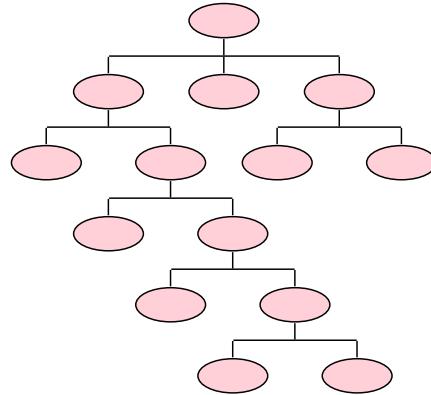
Since there is only one file system, all user files, programs and devices exist in the same name space. By convention, programs and devices exist in their own sub-directories, and users exist in their own sub-directories.

Organisation of Processes

- Processes are organised into a process tree

All processes have a parent process (save the first) and may have child processes.

Each command is executed as a new process, and is the child of the process which invoked it.



Unix does not discriminate between the execution of system related programs and user programs. Both give rise to processes.

The first process in a Unix system is called init and it runs as part of the procedure of making the machine ready for users. init creates child processes which set-up the machine and ultimately prompt the user to login. Once a user has logged in, a new child process is created to enable the user to enter commands. This process is called a shell.

The shell prompts the user to enter a command. For each command that the user enters, the shell spawns a new processes. The shell is therefore the parent of these processes. The shell is itself the child of init, since this gave rise to its creation when the user logged in. Many Unix commands also give rise to child processes, which may in turn give rise to new generations of processes. Thus a tree of processes is seen to have been spawned, starting from the great grand parent of all processes, init.

Login

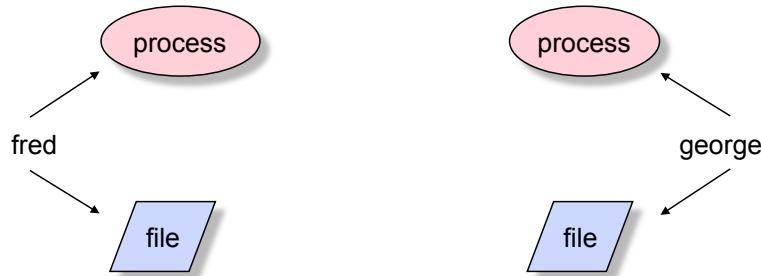
- Accessing Unix/Linux requires users to login
 - validate username
 - associate with ‘account’
- The login process establishes the user’s
 - identity (User and Group)
 - initial directory (home directory)
 - initial process to run (normally a shell)
- Users view Unix through
 - interactive shells
 - custom menus
 - windowing environments
 - applications

In order to use a Unix machine users must login. In this process the system determines who the user is (the username), verifies this information (by requesting a password), and then associates with the user their file and process resources. Specifically, the user is associated with a sub-directory of the file system (their home directory) and an initial process (usually an interactive shell).

Humans tend to use names to distinguish similar objects, machines tend to prefer numbers. During the login process the system associates a UID (user identity number corresponding to the username) and GID (group identity number corresponding to the user's default group) with the user. This is subsequently used to label all files and processes created by the user.

What are Users?

- Users are the owners of files and processes



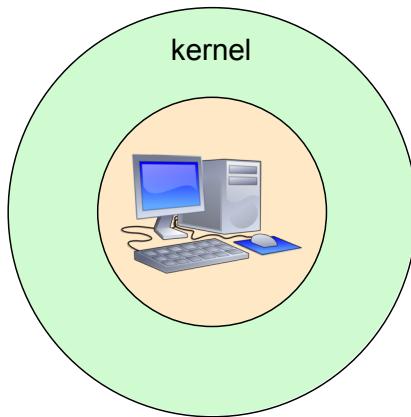
- In Unix everything carries a UID and GID

Users are not first class entities within Unix, only files and processes can claim this status. Users are simply attributes of files and processes. In Unix, every file and every processes must be owned by someone and exist in a group.

When users first login their initial process (the shell) and file (their home directory) belongs to them. Every subsequent file or process created by a user is stamped with the users identity (UID) and the users default group identity (GID). The UID is usually unique and has a one-to-one mapping with the users username; the GID is shared by users working together. The GID provides a means by which users can gain joint access to shared files and commands. In addition to the default GID, every user has up to 16 additional group ids, which are used for access permissions only (not for file ownership).

The Structure of Unix

- A series of layers
- Kernel shields applications from hardware details

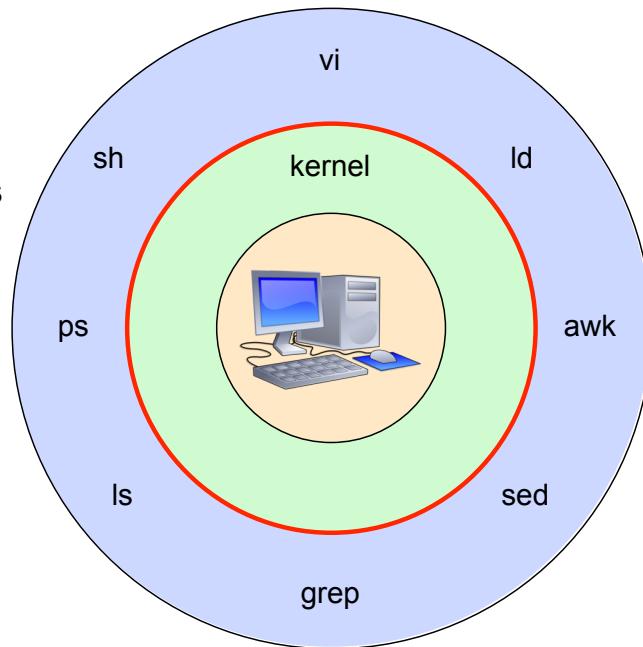


The architectural structure of Unix/Linux is based on a series of layers, each of which provides a more abstract view of the system. The graphical representation of this is often described as the "onion diagram" as it can be imagined as a series of concentric rings.

At the heart of the system, of course, is the hardware. Unix prevents applications from directly accessing this hardware by surrounding it with a low level layer of software known as the kernel. One of the key features of Unix, since the very beginning, has been to provide a consistent view of the features provided by the hardware, while hiding the details of how these features are implemented in specific hardware. This is the basis of how Unix systems and their applications can run on such a wide range of hardware, from mobile devices to supercomputers.

The Structure of Unix

- A series of layers
- Kernel shields applications from hardware details
- User interacts with system through applications



Users interact with a Unix/Linux system through application programs, which perform all sorts of functions from administrative through development support to end user type tasks.

The applications make use of the facilities provided by the system (hardware) through a well defined interface to the kernel, known as the system call interface.

3 Working with Unix/Linux

Accessing Unix Systems 43

Basic Commands 44

Keyboard Control 45

File System Commands 46

Process Commands 48

On-Line Manual 49

Command Format 51

Unix Command Format 52

The Shell 53

Shell Wildcards 54

I/O Redirection 55

Logging out 56

Changing Your Password 57

Accessing Unix Systems

- Most Unix systems at Morgan Stanley are servers
 - located in data centers
- Access is through remote services
 - ssh for terminal access
 - PuTTY for access from Windows desktops
- Hummingbird for remote GUI operation
 - using X window system
- Username and password required for login
 - based on Single Sign On (SSO)

To access a Unix machine a user must first login.

Most of the Unix/Linux systems at Morgan Stanley are servers, located at remote data centers. Access to them is therefore handled across network links.

The most common pattern for Unix/Linux access is to use terminal based command line access based on SSH (Secure Shell). For access to Unix systems from Windows desktops the PuTTY product provides a powerful and flexible way of achieving this, access from one Unix/Linux system to another can be achieved using the built in ssh client application. This course will focus on terminal based access.

It is still possible to run GUI based Unix/Linux applications from the servers to Windows desktops, using the Hummingbird eXceed product that implements support for X Windows based GUI applications on Windows desktops.

In all cases, however, it is necessary to provide a valid Unix username and password during the login process.

Basic Commands

- Simple commands for retrieving basic system information

```
$ date
Wed Jan 27 11:11:33 GMT 1993
$ pwd
/home/fred
$ who am i
sparc2!fred tttyp3 Jan 27 09:28 (:0.0)
$ ls
report errors updates letter
$ who
fred    tttyp3  Jan 27 09:28
prjlg   tttyp4  Jan 26 15:26
roger   tttyp5  Jan 26 17:15
$
```

Users issue commands by simply typing their name into the shell (along with any arguments). The shell will then invoke the corresponding program, giving rise to a process which will carry out the required task.

In the above, date displays the current time and date, pwd (print working directory) shows your current location within the file system, who am i tells you who you logged in as (you may have several usernames), ls lists the files in the current directory and who tells you who is logged in to the machine.

Within Unix there are hundreds of simple commands such as these. In isolation the commands are quite limited, however Unix allows them to be executed in sequence. Complex commands are composed of chains of simple ones, where often the output of one command forms the input to another. Because each command is little more than a building block, their output tends to be terse. In the above their are no headers or footers; for example, to identify the columns of output produced by the who command. This is because the output may form the input of another command, in which case such extra data would be problematic.

Keyboard Control

- Essential control signals

^C	interrupt command
^Z	suspend command
^D	end of file
^H (^?) [DELETE]	delete last character
^W	delete last word
^U	delete line
^S	suspend output (rarely used)
^Q	continue output (rarely used)

When working at the keyboard, Unix/Linux supports a number of "control" characters that allow various operations to be performed.

The above definitions are for the "usual" mapping and may vary from one user to another. Differences occur because of hardware setup and because of the software terminal settings. The stty command is used to establish specific key settings; for example,

```
stty erase <BACKSPACE>
```

makes the backspace key the erase character.

File System Commands

- Moving around the file system

```
$ cd /usr/bin  
$ pwd  
/usr/bin  
$ cd ..  
$ pwd  
/usr  
$ cd  
$ pwd  
/home/fred
```

Unix provides a variety of commands for dealing with the file system. The `cd` command (as adopted by DOS) moves the shell to a specified directory within the logical filesystem. If no arguments are given, then the command locates the shell at the user's home directory. To determine the directory currently occupied by the shell, use the `pwd` (print working directory) command.

The notion of the shell being in a directory deserves some consideration. The shell is a running process. When a user asks the shell to invoke a command, the command is executed with respect to the shell's current directory. That is, the shell process passes to the new command's process its current location. The location is subsequently used by the command whenever it needs to access local files. For example, the `ls` command shows the files in the current directory, by default, because it learns of the current directory from its parent process, the shell.

File System Commands

- Moving around the file system and viewing files

```
$ cd /usr/bin  
$ pwd  
/usr/bin  
$ cd ..  
$ pwd  
/usr  
$ cd  
$ pwd  
/home/fred
```

```
$ ls  
report errors updates letter  
$ cat report  
A Report on Unix Training  
=====  
Once upon a time there was a PDP-7 and a  
space travel game.  
  
$ wc report  
      5      19     113 report  
$ wc -l report  
      5 report
```

The cat command is used to view the contents of a file. In the absence of any other information, cat expects to find the file in the current directory, as established by the shell. cat is an abbreviation for concatenate, meaning to append. It appends the specified file or files to the end of the terminal file.

The terminal file is a special device file associated with the physical terminal device (or workstation window) which the user is using.

wc counts the number of words, lines and characters in the specified files. Using the optional '-l' argument, wc only displays the number of lines in the file.

Process Commands

- Almost all commands give rise to new processes

```
$ more report
A Report on Unix Training
=====
Once upon a time there was a PDP-7 and
a space travel game.

$ cd
$ pwd
/home/fred

$ ps -f
UID  PID PPID C STIME    TTY     TIME CMD
dave 139 101 1 09:42:04 pts/3 0:08 bash
dave 507 139 1 09:51:02 pts/4 0:00 ps -f

$
```

more is invoked as a
new process

cd and pwd are internal
commands and do not
give rise to new processes

Almost all commands issued by users give rise to the creation of a new process. The command interpreter known as the shell has minimal built-in functionality. Its purpose is simply to find the requested program file, invoke it, wait for the process to terminate, and then prompt the user for the next command.

cd and pwd are exceptions to this rule and are built into the shell. The notions of the current directory and of changing the current directory concern the shell itself. Therefore, these commands need to act on the shell directly, not on some child process.

To determine what processes are currently running use the ps command. Using the -u uid (your userid, as listed by the id command) option it displays all processes owned by the user; using the -ef options, it displays all processes on the system. In order to see the size of all programs, use ps -ely.

more is one of many paginating programs in Unix. more displays the contents of files one page at a time. It allows a file to be paged forwards and backwards, and supports searching. Type 'h' within more to determine the available commands.

On-Line Manual

- A comprehensive on-line manual is available for all commands

```
$ man ls
LS(1)                      USER COMMANDS                  LS(1)

NAME
    ls - list the contents of a directory

SYNOPSIS
    ls [ -aAcCdfFgilLqrRstul ] filename ...

AVAILABILITY
    The System V version of this command is available with the
    System V software installation option. Refer to Installing
    SunOS 4.1 for information on how to install optional
    software.

...
```

Most Unix systems are setup with an on-line manual. On current systems this may take several forms, and be accessible through window-based hypertext applications. However, a command line version of the manual should also be available.

The manual pages are stored in the file system within the directory /usr/share/man. Each command has its own manual entry and may be retrieved using the man command. Manual pages are automatically paginated through the more command. It is possible to search a manual entry by using the search commands within more.

On-Line Manual

- A comprehensive on-line manual is available for all commands

```
$ man cat
...
$ man -k print
lpq (1)   - display the queue of printer jobs
lpr (1)   - send a job to the printer
lprm (1)  - remove jobs from the printer queue
pr (1V)   - prepare file(s) for printing
...
$ man -s 1 intro
...
```

Although man provides a means of determining how a command works, it is not very useful if a user is unsure of a command's name. To search the manual pages for all commands which may be suitable for a specified task, perform a keyword search. This is achieved using the -k option. Note that keyword searches only work if the system administrator has properly configured the manual pages using catman, and where appropriate has set up the MANPATH variable.

The manual pages are in fact divided into a series of volumes. Each volume may be directly accessed by supplying a numerical argument from 1 to 8. To determine the nature of each volume, examine the intro entry which exists in each of them.

Command Format

- Commands follow a standard format

```
command [-options] [files ...]
```

- Syntax

- command name, possible options, possible files
- parts of command are separated by white space
- entire command is CASE SENSITIVE

- Philosophy

- each command is simple and designed to do one thing well
- by default commands tend not to generate headers or trailers
- most commands are terse, and take single character options

Unix commands follow a relatively standard format. They consist of the command name, a series of single character options preceded with a dash, and an optional set of files on which the command is to operate.

Unfortunately, due to the somewhat uncontrolled evolution of Unix, this standard is not always adhered to. Some commands do not require a leading dash to indicate an argument string, others require words to identify each option rather than the usual single character. Of perhaps more concern, is the inconsistency which exists between commands. For example, a `-l` option may mean long format to one command and quite the opposite to another.

The various parts of a command are separated by white space, that is, any number spaces, tabs or (where it is apparent to the command) new lines. In general, commands, options and file names are lower case in Unix. The only time that commands or file names exist in upper case, is when it is necessary to emphasise them. For example, some directories may contain a `README` file, drawing the users attention before they interfere with the directory's contents.

Unix is case sensitive. A lower case name is distinct from an upper case name, or indeed a mixed name. Command options are also case sensitive. In fact, where upper case options are available, they often do the opposite to their lower case counterparts.

Unix Command Format

- Listing the files in the /etc directory in long format

```
ls -la /etc
```

- Looking at the contents of three files

```
cat report update letter
```

- Adding line numbers to the displayed files

```
cat -n report update letter
```

The above three examples demonstrate the general format of Unix commands.

In the first example two options are supplied to the ls command. It is clear to ls that -la are options because of the leading dash. Without the dash, ls would attempt to list the contents of a directory called la.

The second example shows the use of the cat command with three filename arguments, but no options. cat, as its name suggests, will concatenate each file in turn and write the composite output to the default output file---the user's terminal. Most Unix commands are able to operate on as many file arguments as are supplied. The commands simply apply themselves to each file in turn.

In the last example an option is supplied to cat requesting that line numbers be generated for the composite output file.

The Shell

- Login shell invoked as the user logs in
 - interactive command line interpreter
 - invokes child processes (commands) for the user
 - is the top of the user's process tree
 - dies when the user logs out
- There are a family of shells available on Unix
 - bourne shell sh \$
 - korn shell ksh \$
 - C shell csh %
 - Bourne Again Shell bash \$
- Shells help users write commands
 - generate lists of filenames
 - redirect command input and output
 - construct command pipelines

Commands which a user submits to Unix are interpreted by the shell. The shell is the process which reads the characters typed in from the keyboard, and eventually invokes the corresponding program. Commands (except for a few minor exceptions) exist as program files somewhere within the logical file system. Part of the job of the shell is to find the location of a requested command, which it does by searching a local variable called its PATH.

There are a number of shells on Unix systems. Since the shell is simply a program, many have been written for Unix as it has evolved. The most important shells are the Bourne shell (the original shell for Unix developed by Steven Bourne), the C shell (the California shell developed at UCB) and the Korn shell (developed by David Korn). The Korn shell is currently the most popular shell.

In addition to invoking commands, the shell can assist users by automatically generating file name argument lists. Using special wildcard symbols, the shell may be requested to complete a command by generating all file names which match the specified wildcard.

The shell also has considerable influence over the default input and output streams used by a child process. By default, processes read input from the keyboard and write output to the display. However, since these are simply files in Unix, the shell is able to change the files which these correspond to. In fact, the shell can even arrange that the output of one command is fed directly into the input of another.

Shell Wildcards

- Wildcards may be used to generate filenames

```
$ ls
f1      f11     f3      two.C
f10     f2      one.C
$ ls f*
f1      f10     f11     f2      f3
$ ls *.C
one.C  two.C
$ ls f?
f1      f2      f3
$ ls f??
f10     f11
$ ls *[13]
f1      f11     f3
```

*	any number of characters
?	any single character
[ab]	a or b or specified range

The shell provides a series of wildcard characters which users may use to ask the shell to automatically complete or generate lists of filenames. In the above examples, * is used to match any number of any character. The ls f* command matches all file names which begin with f followed by any number of any character. ? matches any single character, and [] matches a range of specified characters.

Note that the wildcard mechanism implemented in Unix is different from that used by other operating systems, and in particular from that used by DOS. In DOS individual utilities expand wildcards and therefore interpret them in their own individual ways. In Unix, by contrast, the shell expands the wildcards before passing the generated arguments to commands. Therefore, in Unix individual commands do not have to understand wildcards or implement the supporting code. More importantly, unlike systems such as DOS, the semantics of wildcard symbols in Unix is completely consistent.

I/O Redirection

- The output of a command can be redirected
 - to a file or another command

```
$ ls > new
$ cat new
f1
f10
f11
f2
f3
new
one.c
two.C
$ cat f1 f2 f3 > all
$ ls | wc -l
9
$ ls /etc | more
...
$ echo hello > afile
$ cat afile
hello
```

> redirect output
< redirect input
| pipeline commands

The shell may be used to redirect input and output for commands. By default the input for a command (if not taken from a supplied file) is read from the keyboard (the standard input). Similarly, the output for a command is written to the display (or standard output) by default. The shell understands a number of special characters which direct it to change the files which constitute the standard input or output of a command.

The > indicates that the output should be written to the specified file, the < indicates that the input should be read from a specified file, and the | symbol indicates that the output of the left command should be directed to the input of the right command.

In the first example above the output of ls is stored in a file called new. This is useful if the output needs to be saved or used in another context. In the next example three files are concatenated together by cat, and then appended (one after the other and in sequence) onto the standard output. Due to the shell's redirection, the standard output is the file called all, not the display.

In the next example a command pipeline is established. The output of the ls command is the input of the wc command; with the -l option this outputs the number of files in the current directory. This command sequence is indicative of the power of Unix. Individual commands are not in themselves very powerful. However, by arranging them in pipelines complex tasks can be undertaken. Unix is a powerful toolkit, in which commands interoperate with each other.

__CODEDONE__
__CODE__CODEDONE__
__CODE__CODEDONE__
__CODE__CODEDONE__

Logging out

- Logout with ^D
 - end of input stream ...
- Some shells require logout or exit
 - configured via a shell variable, like `set -o ignoreeof`

In Unix all devices are considered to be files. When the shell is executing it reads its input data from the terminal file. The input data (usually commands) is read until there is no more, i.e., until the end-of-file character is detected. To generate this character from the terminal, type ^D. Note that both the csh and ksh ignore this character if the ignoreeof variable is set. They require logout or exit to terminate.

Changing Your Password

The screenshot shows the Morgan Stanley Enterprise Computing - Password Reset Tool. At the top, there's a navigation bar with links for Morgan Stanley Today, Enterprise Computing, Support, Admin, and Quick Reference Card. Below the navigation bar, there are tabs for Home, Policies, FAQ, and Prod ID. The main content area is divided into two sections: 'Password Reset Tool' on the left and 'Options' on the right.

Password Reset Tool: This section contains instructions for changing individual or production ID passwords and provides a link to 'Password Examples'. It also includes a note about requesting a voicemail password reset via the Voicemail Password Reset Tool.

Options: This section allows users to select platforms for password change and choose a verification method. The 'Reset my password on the following platforms:' section includes checkboxes for All Platforms, Unix/Kerberos, Windows XP (Active Directory), and Mainframe. The 'Verification method:' section includes radio buttons for Current Unix/Kerberos Password, Current Windows XP (Active Directory) Password, Current Mainframe Password, and Personal Information (US based employees only). A 'CONTINUE' button is located at the bottom right of the options panel.

<http://password>

In standard Unix systems, the passwd command allows users to change their passwords. Passwords are the first line of defence against unauthorised access to the machine. They should be chosen carefully and kept secret. Good passwords contain a mixture of alphabetics and numerics. In Unix, they may be of mixed case and contain punctuation characters. Various security enhancement products exist for Unix. For example, Kerberos provides an added security layer. Most Kerberos commands have the same name as the original Unix command, prefixed with a k. For example, kpasswd changes the Kerberos password and ksu (to be discussed later) runs the Kerberos switch user command.

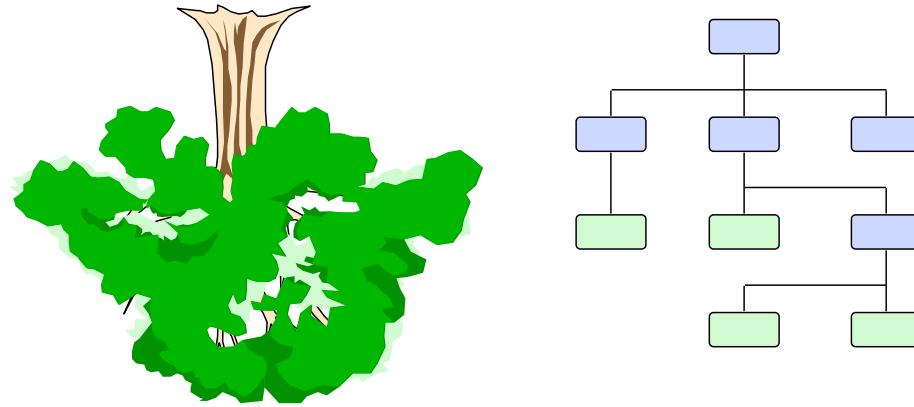
However, at Morgan Stanley the password management infrastructure is specialised so that it can be managed across platforms such as Windows and Unix if necessary. To change your password type password as the URL into a browser and the Enterprise Computing Password Reset Tool is invoked. Passwords changed from here may be directly applied to Unix/Kerberos (email), Windows NT, Windows Active Directory and Mainframe accounts.

4 The File System

- The Unix File System 59
- Hierarchical Structure 60
- Directory Paths 61
- Traversing the File System 62
- Examining Directory Contents 63
- Using Shell Wildcards 65
- Building the File System 66
- Copying Files 67
- Copying with Wildcards 68
- Moving Files 69
- Deleting Files 70
- Linking Files 72
- Hard Links 73
- Symbolic Links 74
- Building the File System Hierarchy 75

The Unix File System

- An organised way of storing files
- The structure of the file system can be thought of as an inverted tree

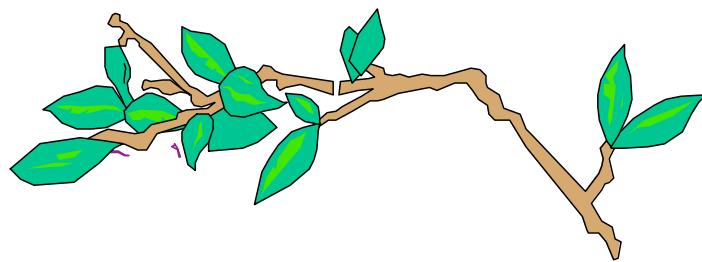


Unix employs a tree structure to store files. Starting from an initial top-level directory (the root directory) sub-directories successively organise information into categories, and then subcategories. There are no limits on the depth to which the tree structure can grow.

Unix differs from other hierarchical file stores (such as those provided in DOS and VMS) in that there is only one tree. The single tree structure hides multiple disks, partitions and even the network when NFS (the Network File System) is employed.

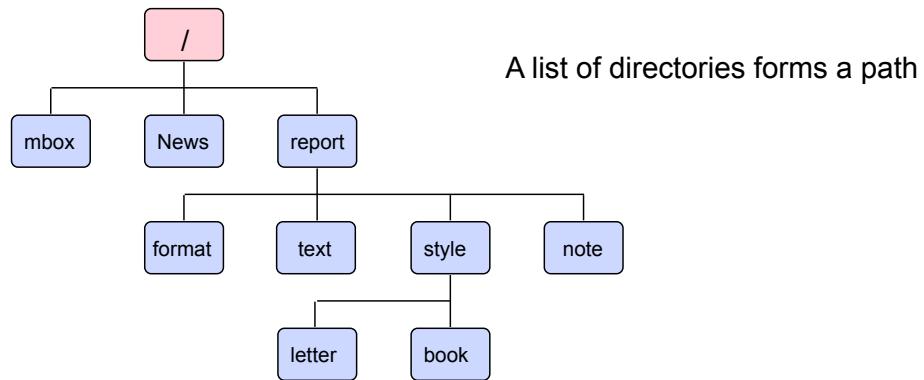
Hierarchical Structure

- Directories are files which hold information on other files
- Directories can be viewed as branches and files as leaves
- Since directories are just other files, they can also be stored inside directories



The Unix file system is organised into a hierarchical tree structure in which directories are branches and files leaves. The purpose of directories is to group together related files. However, since files may themselves be directories, it follows that directories may contain sub-directories.

Directory Paths



- Absolute path names start from root
`/report/style/book`
- Relative path names start from the current directory
`style/book`

Path names describe routes through the file system. A relative path name is a route from the current working directory; an absolute path name is the route from the top of the file system.

Absolute path names begin with a leading slash and are unique. Relative path names are not unique since they depend on the directory in which the path is specified.

Traversing the File System

- Every directory contains two special directory files

“ . ” current directory
“ .. ” parent directory

```
$ cd ../style  
  
$ pwd  
/report/style  
  
$ cd ..  
$ pwd  
/report
```

```
$ cd /report/style  
  
$ pwd  
/report/style  
  
$ cd /report/text  
$ pwd  
/report/text  
  
$ cd .  
$ pwd  
/report/text
```

The directories “.” and “..” are convenient names for the current and parent directories. “..” allows path names to traverse back up through the tree.

Use the `cd` command to ‘change directory’ and the `pwd` command to print the current working directory. Note that `cd` without any arguments takes the user to their home directory.

Examining Directory Contents

- The contents of a directory can be listed using:

```
ls [-aAcCdfFgilLqrRstul] filename ...
```

```
$ pwd  
/report  
$ ls  
format note style  
text  
  
$ ls -F  
format* note style/ text/  
  
$ ls -af  
. format* style/  
../ note text/  
  
$ ls ..  
mbox News report  
  
$ ls /report/style  
book letter
```

The ls command is used to display the contents of the specified directory.

Examining Directory Contents

- The contents of a directory can be listed using:

```
ls [-aAcCdfFgilLqrRstu1] filename ...
```

```
$ ls -l
-rwx----- 1 greg dude 100 Jan 21 20:01 format
-rw----- 1 greg dude 873 Jan 21 19:59 note
drwx----- 2 greg dude 512 Jan 21 19:58 style
drwx----- 2 greg dude 512 Jan 21 19:58 text

$ ls /
mbox News report

$ ls -l style
-rw----- 1 greg dude 22 Jan 21 20:59 book
-rw----- 1 greg dude 230 Jan 21 20:04 letter

$ ls -ld style
drwx----- 2 greg dude 512 Jan 21 19:58 style
```

ls takes a variety of options that affect which files are displayed and the way their information is formatted.

- l long listing
- a all files, including those beginning with '.'
- g used with -l for group ownership
- d the directory file not its contents
- F show file type

Using Shell Wildcards

- Recall shell wildcards for filename expansion

```
$ ls p*
pint    plastered  pub

$ ls /etc/*/*m*
/etc/adm/messages
/etc/dp/modem
/etc/openwin/modules
```

- Note the shell expands wildcards

```
$ echo hello world
hello world

$ echo p*
pint plastered pub
```

The Unix shell provides wildcard expansion to generate filenames for commands. To list all filenames beginning with p (as above), then "*" is used to tell the shell to generate the filenames automatically. The shell searches the specified directory to find the files.

- * matches zero or more characters
- ? matches exactly one character
- [ABC] matches either A or B or C
- [A-Za-z] matches any single letter

Note that wildcard expansion is different in Unix than for DOS. In DOS each individual utility interprets * and may associate a different meaning to the symbol. This is not possible in Unix, since the shell interprets the *, generates an argument list, and then calls the specified command. In the above, echo simply writes to the display its list of arguments; the list of arguments beginning with p were generated by the shell prior to invoking echo.

Building the File System

- `mkdir` creates directories, `rmdir` removes them

```
$ pwd  
/report  
  
$ ls -F  
format*      note      style/  
            text/  
  
$ mkdir biblio  
$ ls -F  
biblio/ note    text/  
format*      style/
```

```
$ rmdir biblio  
$ ls -F  
format*      note      style/  
            text/  
  
$ rmdir style  
rmdir: style: Directory not  
empty
```

```
mkdir [-p] dir1 [dir2 ...]
```

The `mkdir` command is used to create new directories, and `rmdir` to remove directories. In keeping with most Unix commands, the commands may be supplied as many filename arguments as is required. In the following

```
mkdir one two three four five six /tmp/seven
```

six directories are created within the current directory, and a seventh is created beneath `/tmp`. Notice, however, that a minimum of one directory must be supplied to the command.

Using the `-p` option, `mkdir` is able to create missing parent directories as needed

```
mkdir -p first/second/third
```

will create the missing parent directories `first` and `second` if they do not already exist.

Note that it is not possible to remove a directory with `rmdir` if it contains other files. To remove the directory, first remove all the files and sub-directories which it contains. The powerful (and dangerous) `rm -r` command is useful for this.

Copying Files

- cp copies files and directories around the file system

```
$ ls -F  
mbox News/ report/  
  
$ ls -F report/style  
book letter  
  
$ cp report/style/book .  
$ ls -F  
mbox News/ report/ book
```

```
$ ls -F  
mbox News/ report/  
  
$ cp -r report/style .  
$ ls -F  
mbox News/ report/ style/  
  
$ ls -F style  
book letter
```

```
cp [-ip] f1 f2  
cp [-ip] f1 f2 ... fn d  
cp -r [-ip] d1 d2
```

cp is used to copy files and directories around the file system. Note that copy means duplicating the bytes on disk representing the contents of the files being copied.

cp is used with two arguments when copying from one file to another and with many arguments when copying a collection of files into a directory. In the case of the latter, the directory must exist and be the last argument. cp may also be used to copy the contents of one directory to another. In this case the -r (recursive) option must be supplied. When copying directories, if the target (d2) exists, then the source (d1) is created within it. A file f1 within d1 may now also be accessed as d2/d1/f1. If, however, the target does not exist, then it is created and the actual contents of d1 are copied into it. Therefore, a file f1 within d1, may now also be accessed as d2/f1.

By default, the cp command overwrites any files which already exist with the target name. The -i (interactive) option may be used in order to get cp to prompt prior to overwriting any existing files.

To preserve a file's modification time and permission bits, use the -p option. If it is also necessary to preserve the file's ownership, then the cpio command should be used. Some versions of cp support this behaviour directly, such as GNU cp with the -a option.

Copying with Wildcards

- Wildcard may be used when copying collection of files

```
$ cp *.c /home/george/src
```

– if more than two arguments, last one must be a directory

- Beware wildcards when no matching files

```
$ ls
chapter1.txt    chapter2.txt
$ cp *.txt somewhere_else
$ cp *.bak somewhere_else
cp: cannot stat `*.bak': No such file or directory
$ echo *.bak
*.bak
```

Wildcards may be used with the cp command to generate a list of filename arguments to be copied. The cp command operates with different patterns of arguments, however if there are more than two arguments supplied, the last one must be a directory - all the other arguments are copied into this directory.

When using wildcards, we need to be careful to take account of the situation where the pattern does not match any filenames. The behaviour when this occurs is actually dependent on the shell (it is, after all, the shell that expands the wildcards). However the most common result is for the shell to leave the pattern with the wildcards unexpanded. This is legal since the wildcard characters are themselves valid filename characters, however it can lead to unexpected behaviour if we are not expecting it.

Moving Files

- Files and sub-directories can be moved

```
$ ls -F  
mbox News/ report/  
  
$ ls -F report/style  
book letter  
  
$ mv report/style/book .  
$ ls -F  
mbox News/ report/ book  
  
$ ls -F report/style  
letter
```

```
$ ls -F  
mbox News/ report/  
  
$ mv report/style .  
$ ls -F  
mbox News/ report/ style/  
  
$ ls -F style  
book letter
```

```
mv [-i] f1 f2  
mv [-i] f1 f2 ... fn d  
mv [-i] d1 d2
```

mv is used to rename files and directories. It does not cause the contents of the file to be physically moved, only the file's name is changed in its directory.

The new name may be a path to another directory, so mv can in fact move a file or directory from one place to another.

Note that there is no need for a recursive option when moving a directory since files contained within the directory don't care what it is called. More specifically, the contents of a directory file are the files stored within it, and mv does not effect file contents. The -i option may be used if there is a danger of overwriting existing files.

Deleting Files

- **rm** deletes files and directory structures

```
$ ls -F  
mbox  News/  report/  book  
$ rm book  
$ ls -F  
mbox  News/  report/
```

```
$ ls -F  
mbox  News/  report/  style/  
$ rm -r style  
$ ls -F  
mbox  News/  report/
```

The **rm** command deletes files and directories. Beware that in Unix a deleted file is lost forever. There is no mechanism to allow a file to be un-deleted since the disk space associated with the file may immediately be re-used by some other process. To recover a deleted file, the administrator must be asked to restore it from a system backup. It is unlikely that the restored file will contain recent changes made to the file.

The **rm** command also has the **-r** option similar to those of **cp**. The **-r** option is necessary if directory structures must be deleted.

Deleting Files

- `rm` deletes files and directory structures

```
$ ls -F  
mbox News/ report/ book  
$ rm book  
$ ls -F  
mbox News/ report/
```

```
$ ls -F  
mbox News/ report/ style/  
$ rm -r style  
$ ls -F  
mbox News/ report/
```

- `rm -i` requests confirmation before deleting files

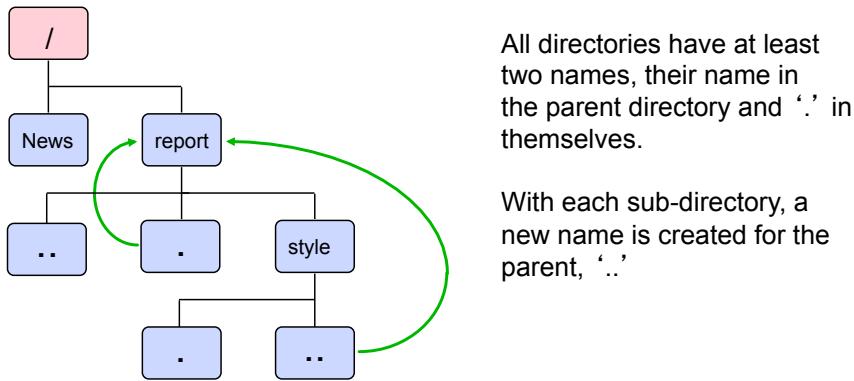
```
$ rm -i book  
rm: remove book? n  
  
$ ls -F  
mbox News/ report/ book
```

```
$ rm -i book  
rm: remove book? y  
  
$ ls -F  
mbox News/ report/
```

The `-i` option allows interactive use of the command so that a user may stop the command from accidentally deleting a file. The `-f` (force) option instructs `rm` to remove files without asking. It would ask if the `-i` option has been set, or if the permissions on the file to be removed are not read/write. In the case of the latter, `rm` will override the permission if the user owns the file.

Linking Files

- Unix directories have multiple names



- It is also possible to create multiple names for files

The Unix file system is held together through links. Each file is identified by a link name, or file name as it is usually called. Every file in the file system must have a name (a link), however it is possible (and often necessary) that some files have multiple links.

The above example shows the multiple names associated with directories. All directories have at least two names, their name in the parent and '.' in themselves. Should they have sub-directories, then a new name is generated for them in each sub-directory, '..'. The '.' and '..' directory links are created automatically when a new subdirectory is made. They are used as a short-hand notation to refer to the current or parent directories.

Hard Links

- Commands have different names to indicate behaviour

```
$ ls -li /usr/ucb/vi  
12334 -rwxr-xr-x 7 root bin 204800 Jul 23 1992 /usr/ucb/vi
```

- Use the ln command to create links

```
$ echo hello > afile  
$ ls -l afile  
-rw-r--r-- 1 fred dude 6 Jan 14 23:16 afile  
$ ln afile newName  
$ ls -l afile newName  
-rw-r--r-- 2 fred dude 6 Jan 14 23:16 afile  
-rw-r--r-- 2 fred dude 6 Jan 14 23:16 newName  
$ cat newName  
hello
```

```
ln [-s] f1 f2
```

It is sometimes useful for normal files to have multiple names. For example, the standard Unix editor, vi, has seven names. The different names for vi include ed, view and vipw. The purpose of the multiple names is to give the illusion that there are many programs providing vi like editing facilities rather than simply one. When invoked, vi checks to see what it was called and changes its behaviour accordingly. This saves users having to remember large combinations of options.

The ln command enables users to create their own links. In the above example, the names newFile and afile are linked to the same file. The fact that afile existed first is not relevant, both are equal. Changes made to the file through the name afile would be the same if made through the file name newFile. The two names refer to exactly the same area on disk.

A useful application of links is to give the illusion that a file exists in multiple directories. This is achieved by specifying path names with the ln command. In the event that the directories physically exist on different partitions, the -s option (for symbolic) must be used to establish the link.

Note that links are destroyed by using the rm command. When the number of links referring to a file is zero, the actual file contents are removed. In fact, the rm command is implemented using the unlink() system call.

Symbolic Links

- Look like hard links but implemented by reference
 - symbolic file contains pathname of file it refers to
 - each has a distinct i-node number

```
$ ls
$ echo hello > file
$ ls -l
total 8
-rw-r--r-- 1 evad  staff  6 22 Aug 17:09 file
$ ln -s file new
$ ls -l
total 16
-rw-r--r-- 1 evad  staff  6 22 Aug 17:09 file
lrwxr-xr-x  1 evad  staff   4 22 Aug 17:09 new -> file
$ uname > new
$ cat file
Darwin
$ rm file
$ cat new
cat: new: No such file or directory
```

File names in Unix are hard-links to files managed in the file system, with the name typically resolving to a specific i-node number via a directory lookup (of the directory file containing the hard-link in question).

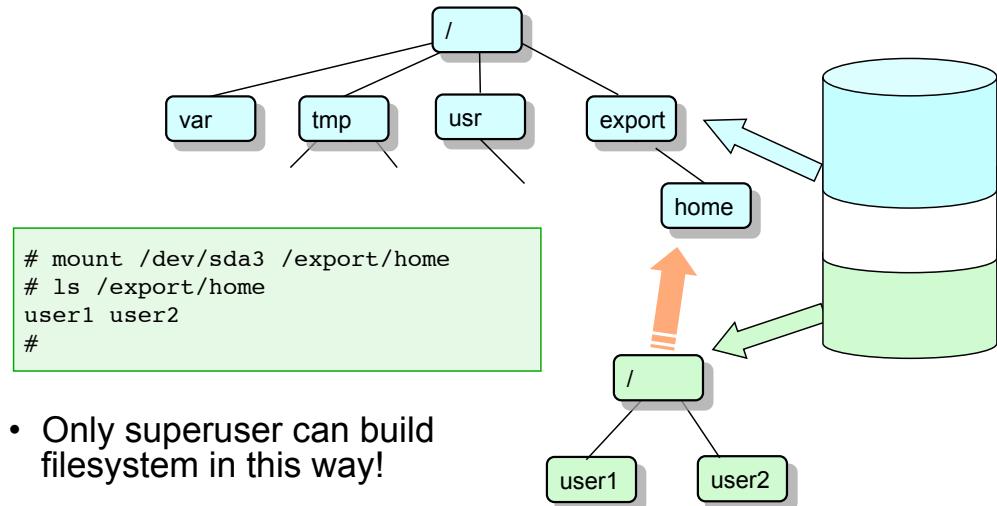
Symbolic links are special files whose content is the pathname of the file to which the link refers. When a file has multiple hard-links, it has multiple names that refer to its i-node (via directory lookup). However, a symbolic link does not refer to a file's i-node, but rather to a new file whose content refers to the file. The file system automatically reads the content of a symbolic link file, and redirects the file I/O to the file whose path is stored inside.

Note that the specific implementation details vary from file system to file system. The original System V file system stored the path as file content, whereas the Berkeley Fat File System stored the file path in the i-node entry for block addresses, provided it would fit. Some file systems do not support symbolic links.

In the above examples, the symbolic link is created with `ln -s`. The long listing shows that `new` points to `file`, and also shows the file type as `l` (far left hand side). The permissions on the link are of little consequence, as a user can always go directly to the file being referenced. Removing `file` caused the link to fail, as the pathname it contains no longer refers to an existing file. (If this was a hard-link, then removing `file` would simply have decremented the link count on the underlying i-node, and `new` would still be valid.)

Building the File System Hierarchy

- Each disk device has its own hierarchical file system
 - joined together into one hierarchy with mount command



- Only superuser can build filesystem in this way!

Each physical disk on a Unix system is normally divided into a number of "slices", or "virtual disks", which contain their own hierarchical file system. To build the single filesystem hierarchy that the user will see, the filesystems on the disk slices are combined together.

One of the filesystems on the partitions is treated specially, as the root filesystem. Other filesystems are attached to this by a procedure known as mounting.

The filesystem on a particular partition is connected to a directory, known as a mount point. After this, any reference to the directory in a pathname will cause the kernel to automatically cross the mount point and retrieve information from the other disk. Further filesystems may be mounted on directories in the "sub-filesystems".

If the mount point directory contains any files or other subdirectories before the mount operation, these will be temporarily hidden afterwards. The contents are unchanged, but may not be accessed until the filesystem that is hiding them is unmounted.

It is not possible to unmount a filesystem when any files contained in the filesystem are open. This includes any directories being the "current directory" of any processes.

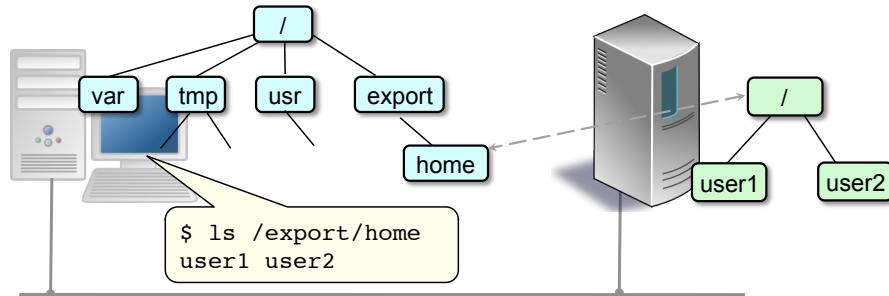
Mounting and unmounting filesystems is a critical operation and may only be performed by the superuser. Most mount operations are performed while the system is booting, and most unmount operations take place when the system shuts down.

5 Network File Systems

- Accessing Files on Remote Systems 77
- Distributed File Systems 78
- AFS at Morgan Stanley 79
- The Morgan Stanley AFS Namespace 80
- Building the Global Virtual Cell 81
- Resolving Symbolic Links 82
- Shared Access to NAS Storage 83
- Recovering Deleted Files 86

Accessing Files on Remote Systems

- Filesystem hierarchy can span systems
 - as well as disks



- Files on server appear local to client machines
 - clients are unaware of the actual location of files

The mechanism for combining file hierarchies from different disks on one machine is easily extended to allow file hierarchies from different machines to be combined, so that one Unix system presents a single unified hierarchy of files and directories to users. The physical location of the files is not known to users - it may be on the local system or a remote system.

In Morgan Stanley the Unix environment is highly sophisticated, making use of distributed file systems on many different machines to give all Unix users a consistent hierarchical view of directories and files.

Distributed File Systems

- NFS
 - originally developed by Sun Microsystems
 - extension of local Unix filesystem ideas to network
 - allows access from Windows clients
 - early versions most suitable for small/medium networks
 - WAN performance not good in early versions, resolved in NFS 4
- AFS
 - developed at Carnegie Mellon University
 - aimed to address scalability and WAN issues of NFS
 - does not preserve semantics of Unix files
 - transparent support in Windows and Unix

AFS at Morgan Stanley

- Used in the Unix & Windows Environment
 - /ms/... top of AFS global file system
 - /ms/user/
 - User home directories
 - /ms/dev /ms/dist
 - Software development and distribution
 - access available from Unix and Windows systems
- Read-only access to AFS files is available via the Intranet

The screenshot shows a Microsoft Internet Explorer window displaying a file index. The title bar reads "Index of /ms/user/e/ewsmith - Microsoft Internet Explorer provided by Morgan Stanley". The address bar shows the URL "http://afsweb.ms.com/ms/user/e/ewsmith". The page content is titled "Index of /ms/user/e/ewsmith" and lists various files and directories with their last modified date and size. The table has columns for Name, Last modified, and Size.

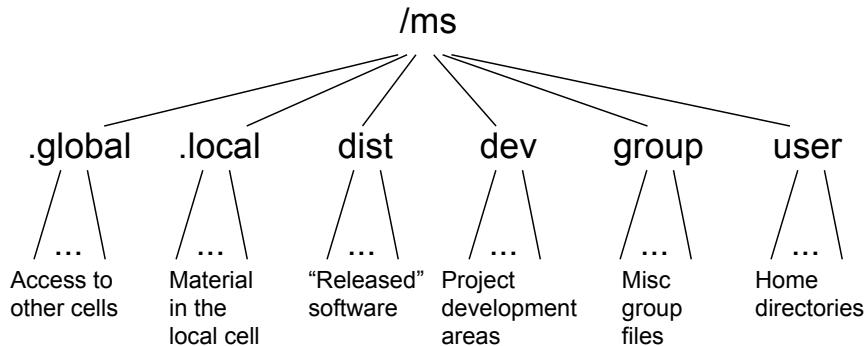
Name	Last modified	Size
Parent Directory	17-Mar-2010 12:44	-
1.k	01-Feb-2010 06:33	1k
Appendix.ppt	16-May-2008 02:37	101k
Autosys.pdf	16-May-2008 02:37	59k
XML_Overview/	06-Nov-2006 02:47	-
scls/	11-Feb-2010 01:21	-
ssses/	11-Sep-2009 23:55	-
autosys-06-06-2005.pdf	16-May-2008 02:37	493k
bin/	10-Feb-2010 01:34	-
class-ewsmith-erpjob.jil	16-May-2008 02:37	1k
class-ewsmith-tutorial.jil	16-May-2008 02:37	1k
core/	03-Feb-2010 23:01	-
development2.ppt	16-May-2008 02:37	429k
edb_scripts/	02-Dec-2007 01:00	-
evolution/	10-Oct-2006 13:57	-
java/	27-Sep-2007 07:49	-
kerberos.pdf	16-May-2008 02:37	35k
list_dhcp_servers	16-May-2008 02:37	1k
makefile	28-Aug-2009 23:40	1k
makefile-c	29-Aug-2009 00:36	1k
makefile-ccpp	29-Aug-2009 00:19	1k
message	16-May-2008 02:37	4k
..../	11-Dec-2009 19:16	-

AFS is a global shared file system deployed (and extensively re-engineered) at Morgan Stanley. Any files under the /ms directory are being managed by AFS.

We can access AFS files from Unix or Windows systems, additionally there is read-only web based access provided on the Intranet.

The Morgan Stanley AFS Namespace

- One top-level AFS “mount point”
 - /ms instead of /afs
- Common “canonical” namespace underneath:



In a proper globally distributed filesystem, users should be able to access the same set of files and directories from any system on the network, using a consistent naming system. AFS gives this capability within a given cell, however at MS the principle is extended to provide a global namespace that hides from users the underlying structures of the many cells that go together to make up the Aurora filesystem.

As this is not a conventional AFS installation, Aurora uses a different directory as the “root” of the AFS namespace, /ms instead of /afs. Next, the second level of directory underneath the AFS root is not named according to cell name. The aim is to provide a “Global Virtual Cell”, made up from many other cells. The underlying structure should be transparent to users, to provide location transparency for files and directories.

So a common set of directories is enabled underneath the AFS root. The four main ones, dev, dist, group and user, provide access to the information while the other two, .global and .local, provide the means by which all is glued together. Since .global and .local are not usually displayed by tools such as ls, and shell wildcarding, the connection mechanisms are in effect hidden from users. They will see:

dev - where all development projects have their storage. The organisation of this area is described later in this chapter.

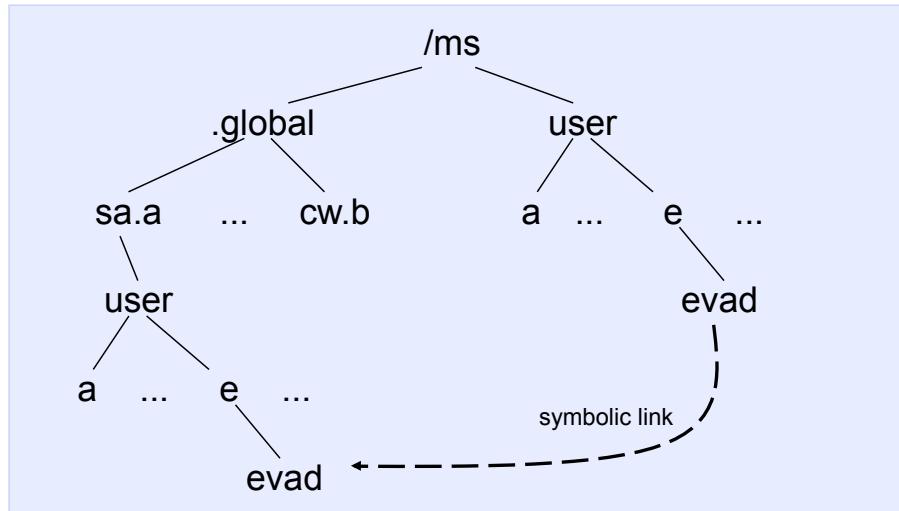
dist - software that has been “released” and is available for use by users. This may be from internal projects, or may be standard or third party applications. This is the only area that is replicated.

group- miscellaneous group related areas.

user - home directories

Building the Global Virtual Cell

- Cells are “glued together” underneath `/ms/.global`
- Symbolic links connect standard names into this structure:



The directory `/ms/.global` presents a common point where the various cells can be connected together, giving what appears to be one large, global cell.

Underneath this directory, there is one directory for every cell, named according to the cell name. Each cell directory contains four further subdirectories, named according to the standard naming conventions for the MS AFS namespace, ie. dev, dist, user and group. In each of these directories can be found the data provided to the global AFS namespace by that cell.

Users access the data through the four main directories; symbolic links are used to connect to the AFS data from the appropriate cell.

For example, in the slide we see the pathname for the home directory of the user evad. MS organises home directories so they appear under the directory `/ms/home`. Because of the many thousands of users who must be managed in this way, a second level of directory was inserted, organised according to the first character of the users' login names. So we have `/ms/user/a/...` up to `/ms/user/z/...` Our user, evad, will have a home directory `/ms/user/e/evad`.

The home directory is a symbolic link to the directory in the appropriate `/ms/.global` directory that acts as a mount point for the AFS volume that contains the actual home directory. So the user sees only the innocent directory name `/ms/user/e/evad`, unaware that accessing this may involve many AFS transactions.

Resolving Symbolic Links

- Morgan Stanley environment relies on symbolic links
 - often deeply nested
 - ls command only shows immediate link target
 - sl command resolves links to ultimate target

```
$ ls -l /ms/dist/aurora/bin/sl
lrwxr-xr-x 1 afsadmin root 31 Nov 21 1996 /ms/dist/aurora/bin/sl
-> ../../PROJ/misc/incr/common/bin/sl

$ sl -v /ms/dist/aurora/bin/sl
/ms/dist/aurora/bin/sl:
/ms/dist/aurora/bin/sl -> ../../PROJ/misc/incr/common/bin/sl
    PROJ/misc/incr/common/bin/sl ->
        /ms/dist/aurora/PROJ/filetools/prod/bin/sl
/ms/dist/aurora/PROJ/filetools/prod -> 1.5
    1.5/bin/sl -> ../../common/bin/sl
    common/bin/sl
/ms/dist/aurora/PROJ/filetools/1.5/common/bin/sl
```

The Morgan Stanley Unix environment depends heavily on the use of symbolic links. Often these links are deeply nested, and it is not unusual for there to be more than ten links in the hierarchy to the target file.

There is an in-house developed tool called sl, which will resolve, and report, all the symbolic links to the ultimate target.

Shared Access to NAS Storage

- Network Attached Storage (NAS) devices are accessible
 - via NFS from Unix systems
 - via CIFS from Windows systems
- Users' personal storage space
 - accessible via /v path on a Unix system
 - accessible via U: drive on a Windows system
- The same physical storage
 - just accessed via different protocols

The NAS storage systems in use by the Firm support access to their storage via both NFS (Network File System) from the Unix environment and CIFS (Common Internet File System) from the Windows environment.

A user's Windows desktop machine has access to dedicated storage for that user in the U: network drive.

Similarly, a user with a Unix account has a /v path to dedicated storage for that user (which for newer users in the Firm, is also their home directory).

For a given user, both of these apparently different storage areas are, in fact, the same.

The reality is that both of these methods access the same underlying storage. The only difference between them is the protocol used to access the storage by the operating systems.

Shared Access to NAS Storage

- Via NFS on a Unix system:

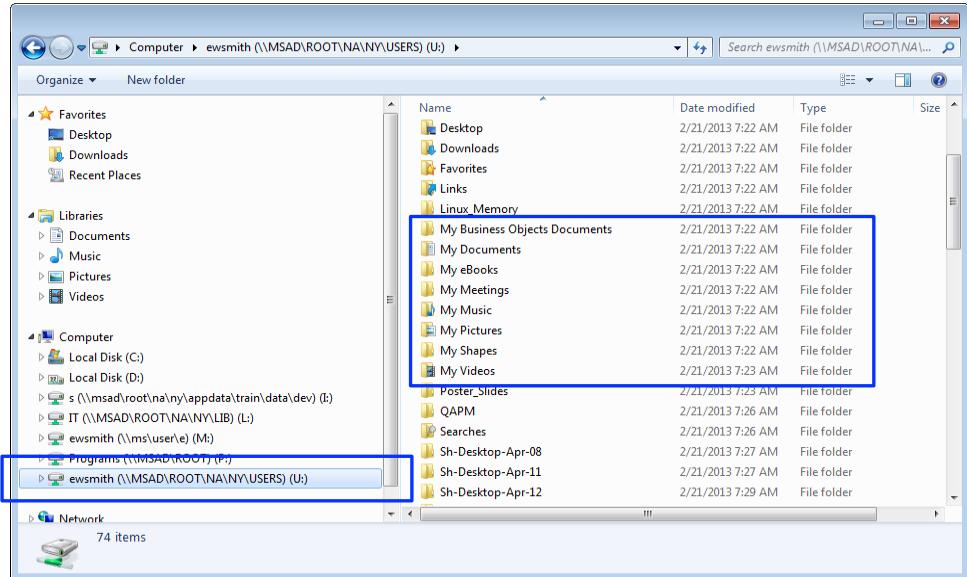
```
$ cd /v/global/user/e/ew/ewsmith
$ ls
...
Linux_Memory
MSCL_StyleGuide_122206.pdf
My Business Objects Documents
My Documents
My Meetings
My Music
My Pictures
My Shapes
My Videos
My eBooks
...
aurora-part1-slides.pdf
desktop.ini
dsdb-partitons.txt
einhop1-slides.pdf
flow-eq-deriv-overview.pptx
heartbeats.txt
null
p4tx
perl-slides.pdf
profilehome
```

- the user's directory (ewsmith) is automounted
- the directory and its content become available when accessed

The /v/global/user/e/ew/ewsmith path on the Unix platform provides NFS access to the shared storage for the user ewsmith on the Unix platform.

Shared Access to NAS Storage

- Via CIFS on a Windows system:



Using the U: drive on a Windows desktop provides access via CIFS to exactly the same storage as that used in the /v/global/user/e/ew/ewsmith path on the Unix platform.

This architecture allows you to create / manage files in the Unix environment, and then manipulate / print them from the Window environment, if so desired, or vice versa.

Recovering Deleted Files

- The Unix philosophy on deleting files
 - once they are deleted, they are gone for good
- Windows moves deleted files to the Recycle Bin
 - it's just a folder, so files aren't really deleted
 - emptying the Recycle Bin really deletes them
 - Unix graphical desktops follow the same principle
- Users cannot recover files
 - that were deleted from the Unix command line
 - once the Recycle Bin has been emptied
- NAS storage infrastructure provides read-only snapshots
 - taken hourly, nightly and weekly
 - permits copying files from the snapshot back to live filesystem

When working from the command line in the Unix environment, the philosophy regarding deleted files has always been that the user knows best, and if the user uses the rm command to delete a file, then that was their intention. Of course, sometimes people do delete files in error.

In graphical user interface environments, such as Windows, the X-Windowing system on Unix / Linux, and the Mac OS X environment, deleting a file in reality merely moves the file to a folder called Recycle Bin, or Wastebasket, or Trash, etc. It is not until the Recycle Bin is emptied, that the file is really deleted. Again, it is entirely possible for files to be deleted unintentionally.

If you are storing files in your /v directory path, or on your Windows U: drive, there is a mechanism to be able to recover files that have been deleted. Snapshots of the filesystem are taken at regular intervals, and assuming that you deleted a file after a snapshot was taken of the file system, or before the last snapshot that contains it is removed, you can recover files.

The snapshot is achieved by taking read-only copies of the inodes in the file system, and so, it cannot be changed (either adding or deleting files).

Recovering Deleted Files

- From a Unix command line
 - in a users' home directory

```
$ pwd  
/v/global/user/e/ew/ewsmith/  
$ cd .snapshot  
$ ls  
hourly.0          nightly.17      nightly.36  
hourly.1          nightly.18      nightly.37  
hourly.10         nightly.19      nightly.38  
hourly.11         nightly.2       nightly.39  
hourly.2          nightly.20      nightly.4  
hourly.3          nightly.21      nightly.40  
...  
$ cd hourly.0 #the most recent snapshot  
$ ls  
0908-Dist-PlatformPrint-Actual.doc      QAPM  
13C-Layout.xlsx                         RECYCLER  
AFS security.doc                        Searches  
Advanced Perl                           Session.piz  
...
```

Within any directory under the user's directory (/v/global/user/a/au/auser), the user can cd to a directory called .snapshot.

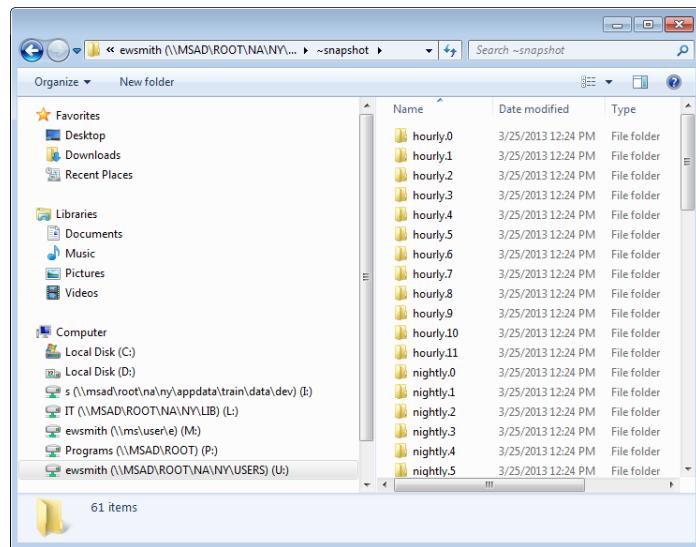
This directory is not normally available, and is not displayed when using the ls command (even with the -a option) because it is mounted on demand, as the user cd's into the directory.

In the directory are further directories for hourly (for the last 12 hours), nightly (for the last 42 days) and weekly (for the last 6 weeks) snapshots of the file system. By cd'ing into one of these directories, there are read-only copies of the files in the filesystem (when the snapshot was taken), which can be copied back to the live file system (outside of the .snapshot hierarchy)

The most recent snapshot for each type has the number 0 at the end of its name.

Recovering Deleted Files

- From a Windows Desktop
 - on the U: drive



All of the snapshots are also available via Windows Explorer, simply by typing .snapshot into the navigation bar, at the end of the current directory (which is being displayed).

After changing to the folder containing the appropriate snapshot, the files in the snapshot are available to be copied and pasted back into the live filesystem.

6 Working with Files

- Unix Files 90
- File Names 91
- Displaying File Contents 92
- Displaying Parts of Files 93
- Searching Files 95
- Quoting 98
- More grep Examples 99
- Sorting Files 100
- Options For Sorting Files 101
- Command Pipelines 102
- Comparing Files 103
- Cutting Fields 104
- Pasting File Contents 105
- Duplicate Lines 106
- Non-ASCII Files 107
- Counting Words 109
- Printing Files 110

Unix Files

- Stored within the Unix file system
- Streams of bytes without structure
- Contain anything: data, text, executable code
- Directories are files containing references to the files stored inside them
- Devices are logical files accessed through the file system

Files are stored within the Unix file system and contain streams of bytes without structure. That is, notions of records, fields and padding do not mean anything to Unix. Such high-level structuring is layered onto the system by application programs.

Files may contain data, text or executable code. Directories are special files which contain references to the files stored within them, specifically they contain filenames and file i-node numbers (the system's name for the file).

Since devices are accessed as files within Unix, they too have filenames. Usually (though not necessarily) device files reside in the /dev directory. Device files can be opened, read, written and closed just like ordinary files. However, transactions with a device file are mapped by the Unix kernel to the physical device. A long listing of a device file shows (in place of size) the index in the kernel of the device driver software controlling the physical device.

File Names

- Files are referenced by filenames
 - relative to the directory in which they are stored
- File names may contain most characters
 - including white space
 - including non-printable ASCII characters
- Maximum filename length is normally 256 characters
 - depends on specific file system implementation
- By convention most filenames are lower case
 - all Unix/Linux operations are case sensitive

Most Unix systems allow up to 256 characters to be used in a filename. The name may consist of almost any character (including spaces and non-printable characters). However, a slash / may not be used as it separates the components of a path name.

By convention, filenames are only in capitals when it is necessary to draw attention to them (README, for example). Most files and indeed virtually all commands use low-case characters.

Displaying File Contents

- cat is used to display contents of files

```
$ cat file1  
This is the only line in file1  
  
$ cat file1 file2 file3  
This is the only line in file1  
This is the only line in file2  
This is the only line in file3
```

- more may be used to paginate contents

```
$ more long_file  
This is the first line  
This is the second line  
.  
.  
.  
This is the twenty third line  
--More--(33%)
```

cat is used to display the contents of a file. cat abbreviates concatenate, meaning 'append'. By default, cat is used to append files in the file system to the end of the device file representing the display.

Note that when supplied with multiple file arguments, cat applies itself to each, one after the other. The files are thereby concatenated one after the other on the display (standard output).

cat will obviously work best when the file contains text - some versions of Unix/Linux allow binary files to be displayed in this way as well, however the output will be rather hard to interpret!

more is useful when the file to be displayed is greater than a single screen. Using more, the file can be viewed page by page. To see the next page of output type <space>, to see the next line type <return>. more shows in the bottom left corner the percentage of the file which has been seen. Type "h" within more to discover the other functions it can perform - there are many!!

Displaying Parts of Files

- head displays the first lines of a file

- defaults to first 10 lines
- argument allows different number to be shown

```
$ cat userlist
dpm      console Jan 23
roger    ttyp0   Jan 23 (csgi55)
csc5gf   ttyp1   Jan 19 (csgi05)
csc6eq   ttyp2   Jan 22 (csgi47)
sta4rf   ttyp3   Jan 23 (csun02)
james    ttyp4   Jan 23
james    ttyp5   Jan 22 (csp0a)
robert   ttyp6   Jan 21 (csp0a)
admis   ttyp7   Jan 23 (suna)
dpm     ttyp8   Jan 23
prj1gf   ttyp9   Jan 22 (blobby)
prjdpm  tttypa  Jan 20 (blobby)
simon   ttypb   Jan 21 (csgi32)
janet   ttyq1   Jan 22
prj1sb  ttyq2   Jan 23
prj3js  tttyt5  Jan 23 (csparc)
```

```
$ head userlist
dpm      console Jan 23
roger    ttyp0   Jan 23 (csgi55)
csc5gf   ttyp1   Jan 19 (csgi05)
csc6eq   ttyp2   Jan 22 (csgi47)
sta4rf   ttyp3   Jan 23 (csun02)
james    ttyp4   Jan 23
james    ttyp5   Jan 22 (csp0a)
robert   ttyp6   Jan 21 (csp0a)
admis   ttyp7   Jan 23 (suna)
dpm     ttyp8   Jan 23

$ head -4 userlist
dpm      console Jan 23
roger    ttyp0   Jan 23 (csgi55)
csc5gf   ttyp1   Jan 19 (csgi05)
csc6eq   ttyp2   Jan 22 (csgi47)
```

The head command enables the first n lines of a file to be concatenated onto the standard output, usually the display. In the absence of the option selecting the number of lines, 10 lines are output.

head is particularly useful when the user is only interested in the first few lines of a large file. It is also useful in command pipelines, selecting the top ten (or so) lines from the output of the preceding command. For example, selecting only the largest ten files in a directory listing.

Displaying Parts of Files

- **tail** displays the last lines of a file
 - shows 10 by default, or specified number

```
$ cat userlist
dpm      console Jan 23
roger    tttyp0   Jan 23 (csgi55)
csc5gf   tttyp1   Jan 19 (csgi05)
csc6eq   tttyp2   Jan 22 (csgi47)
sta4rf   tttyp3   Jan 23 (csun02)
james    tttyp4   Jan 23
james    tttyp5   Jan 22 (csp0a)
robert   tttyp6   Jan 21 (csp0a)
admis   tttyp7   Jan 23 (suna)
dpm     tttyp8   Jan 23
```

```
$ tail -3 userlist
robert  tttyp6   Jan 21 (csp0a)
admis   tttyp7   Jan 23 (suna)
dpm     tttyp8   Jan 23

$ tail +6 userlist
james   tttyp4   Jan 23
james   tttyp5   Jan 22 (csp0a)
robert  tttyp6   Jan 21 (csp0a)
admis   tttyp7   Jan 23 (suna)
dpm     tttyp8   Jan 23
```

```
tail +|-number [-f] [files ...]
```

The tail command displays the last n lines in a file, the last 10 lines by default. Like head, it is useful when dealing with large files since it allows only the portion of the file of interest to be displayed.

tail is more powerful than head because it is able to output the end of the file relative to the start or the end. Using -n tail operates relative to the end, using +n it operates relative to the start.

tail takes another useful argument which enables it to monitor files as they grow. Using -f tail outputs the end of the file and then waits. As the file grows, due to other processes writing to it, tail displays the new lines. This is particularly useful if it is necessary to watch log files as they grow.

Searching Files

- grep searches files for strings

```
$ cat userlist
dpm      console Jan 23
roger    tttyp0   Jan 23 (csgi55)
csc5gf  tttyp1   Jan 19 (csgi05)
csc6eq  tttyp2   Jan 22 (csgi47)
sta4rf  tttyp3   Jan 23 (csun02)
james   tttyp4   Jan 23
james   tttyp3   Jan 22 (csc0a)
robert  tttyp6   Jan 21 (csp0a)
admis   tttyp7   Jan 23 (suna)
dpm     tttyp8   Jan 23
```

```
$ grep csc userlist
csc5gf tttyp1 Jan 19 (csgi05)
csc6eq tttyp2 Jan 22 (csgi47)
james  tttyp3 Jan 22 (csc0a)
```

```
grep [-cilnvw] <RE> [files ...]
```

- Regular Expressions (RE) are string templates

grep is used to search the specified file (or files, or standard input) for all lines which contain the specified regular expression (pattern) and then to print them.

grep is a member of a family of commands to search files. egrep is extended grep and understands a bigger template language, fgrep is fixed grep and does not understand regular expressions at all.

Searching Files

- grep options

- c display count of matching lines
- i case insensitive
- l list names of files containing matching lines
- n precede each line by its line number
- v only display lines that do not match
- w search for the expression as a word

```
$ cat userlist
dpm      console Jan 23
roger    tttyp0   Jan 23 (csgi5)
csc5gf   tttyp1   Jan 19 (csgi05)
csc6eq   tttyp2   Jan 22 (csgi47)
sta4rf   tttyp3   Jan 23 (csun0a)
james    tttyp4   Jan 23
james    tttyp5   Jan 22 (csc0a)
robert   tttyp6   Jan 21 (csp0a)
admis   tttyp7   Jan 23 (suna)
dpm     tttyp8   Jan 23
prj1gf  tttyp9   Jan 22 (blobby)

$ grep -c csc userlist
3

$ grep -n csc userlist
3:csc5gf      tttyp1   Jan 19 (csgi05)
4:csc6eq      tttyp2   Jan 22 (csgi47)
7:james tttyp5 Jan 22 (csc0a)

$ grep -v tty userlist
dpm      console Jan 23
```

grep takes a number of options to modify its behaviour. Options make it case insensitive, count matching lines and precede lines by their line number.

The -l option is useful when searching several files (perhaps hundreds) for a file which contains some known entry. Using -l grep will display the filename in which the match is found.

```
grep -l string *
```

The -v option is useful to negate the search pattern. Using this option all lines which do not contain the pattern are printed. Sometimes it is easier to use a regular expression to describe what is not wanted, rather than what is wanted.

The -w option is useful for looking for regular expressions that describe complete words; especially useful if the expression may be at the beginning, end or middle of the line. Note that \< ... \> is more generally used to define word boundaries.

Searching Files

- Regular expressions are templates for strings

- match any character
- match zero or more occurrences of previous character
- match beginning of line
- match end of line
- [abc] match any one of a, b or c
- [a-z] match any character in range

```
$ cat userlist
dpm      console Jan 23
roger    tttyp0   Jan 23 (csgi55)
csc5gf  tttyp1   Jan 19 (csgi05)
csc6eq  tttyp2   Jan 22 (csgi47)
sta4rf  tttyp3   Jan 23 (csun02)
james    tttyp4   Jan 23
james    tttyp5   Jan 22 (csc0a)
robert   tttyp6   Jan 21 (csp0a)
admis   tttyp7   Jan 23 (suna)
dpm     tttyp8   Jan 23
prj1gf  tttyp9   Jan 22 (blobby)
```

```
$ grep '^csc' userlist
csc5gf tttyp1 Jan 19 (csgi05)
csc6eq tttyp2 Jan 22 (csgi47)

$ grep -v '$' userlist
dpm      console Jan 23
james    tttyp4 Jan 23
dpm     tttyp8 Jan 23

$ grep ro.er userlist
roger   tttyp0 Jan 23 (csgi55)
robert  tttyp6 Jan 21 (dosh$1)
```

Regular expressions provide templates for the strings being matched. Rather than looking for the string anywhere on the line, a regular expression allows context information to be included also. For example, looking for all lines which begin or end with a specified string.

Note that the * in a regular expression is quite different (and more flexible) than that used by the shell for filename wildcards. In the shell * means any number of any character, whilst in regular expressions it means any number of the preceding character.

The single quotes placed around the regular expressions in the above examples, insulate the enclosed characters from the shell.

Quoting

- Quoting allows characters to be hidden from the shell

```
$ echo hello      world
hello world
$ echo "hello      world"
hello      world
$ echo p*
pub pint pissed
$ echo "p*"
p*
```

“...” quotes spaces & wildcards
‘...’ also quotes \$, ^, ”
\. quotes next character

- backslash quotes regular expression characters in grep

The shell provides a number of mechanisms to quote or escape special characters. The quoting mechanisms insulate the characters from the shell, and allow them to be passed directly to the program being invoked.

For example, the * character is used by the shell and by programs which understand regular expressions. Since the shell reads the command line first, it will carry out file name expansion, even though the * may actually have been intended as a regular expression for the program being invoked. Quoting gives the user the opportunity to tell the shell to leave the special characters alone.

There are a variety of special characters used by the shell. *, ? and [] are the standard wildcards and may be escaped using "" double-quotes. \$ is used in variable substitution and ^ in the C-shells command line history mechanism. These must be escaped using the single-quotes ', which may also be used for wildcards.

The backslash \ escapes only the following character but is as powerful as single-quotes. In addition, for the C-shell it escapes !. The backslash may also be passed to programs interpreting regular expressions as a quote. For example, allowing grep to interpret . as a literal character.

More grep Examples

```
$ cat userlist
dpm      console Jan 23
roger    tttyp0   Jan 23 (csgi55)
csc5gf   tttyp1   Jan 19 (csgi05)
csc6eq   tttyp2   Jan 22 (csgi47)
sta4rf   tttyp3   Jan 23 (csun02)
james    tttyp4   Jan 23
james    tttyp5   Jan 22 (csc$a)
robert   tttyp6   Jan 21 (dosh$1)
admis    tttyp7   Jan 23 (suna)
dpm      tttyp8   Jan 23
prj1qf   tttyp9   Jan 22 (blobby)
prjdpm   tttypa  Jan 20 (blobby)
simon   tttypb  Jan 21 (csgi32)
janet   ttyq1    Jan 22
prj1sb   ttყy2   Jan 23
prj3js   ttყt5   Jan 23 (csparc)
```

```
$ grep '(csc.*)' userlist
james   tttyp5   Jan 22 (csc$a)

$ grep 'tty[qts]' userlist
janet   ttyq1   Jan 22
prj1sb   ttყy2   Jan 23
prj3js   ttყt5   Jan 23 (csparc)

$ grep 'tty.[0-2]' userlist
roger   tttyp0   Jan 23 (csgi55)
csc5gf   tttyp1   Jan 19 (csgi05)
csc6eq   tttyp2   Jan 22 (csgi47)
janet   ttyq1   Jan 22
prj1sb   ttყy2   Jan 23

$ grep '\$' userlist
james   tttyp5   Jan 22 (csc$a)
robert   tttyp6   Jan 21 (dosh$1)
```

The first example displays all lines inside the file userlist that contain (csc followed by any number of any characters followed by).

The next example displays all lines that contain tty followed by either q or t or s.

The third example displays all lines which contain tty followed by any single character, followed by a character in the range 0 to 2. Note that the range must be a positive range through the character set.

Quoting from grep

Just as it is sometimes necessary to insulate characters from the shell, it may be necessary to hide characters from applications invoked from the shell. grep uses the \ to escape characters. In the last example, \$ is escaped from both the shell and grep so that all lines containing a literal \$ character are printed.

Sorting Files

- sort organises files into alpha-numeric order
 - default ordering is increasing alphabetic order
 - entire line used as key

```
$ cat userlist
dpm      console Jan 23
roger    tttyp0   Jan 23 (csgi55)
csc5gf   tttyp1   Jan 19 (csgi05)
csc6eq   tttyp2   Jan 22 (csgi47)
sta4rf   tttyp3   Jan 23 (csun02)
james    tttyp4   Jan 23
james    tttyp5   Jan 22 (csc$a)
robert   tttyp6   Jan 21 (dosh$1)
admis   tttyp7   Jan 23 (suna)
dpm     tttyp8   Jan 23
prj1gf   tttyp9   Jan 22 (blobby)
```

```
$ sort userlist
admis   tttyp7   Jan 23 (suna)
csc5gf   tttyp1   Jan 19 (csgi05)
csc6eq   tttyp2   Jan 22 (csgi47)
dpm     tttyp8   Jan 23
dpm     tttyp9   Jan 22 (blobby)
james    tttyp4   Jan 23
james    tttyp5   Jan 22 (csc$a)
prj1gf   tttyp6   Jan 21 (dosh$1)
robert   tttyp7   Jan 23 (csgi55)
sta4rf   tttyp3   Jan 23 (csun02)
```

sort organises the lines in its input files into alphanumeric order. By default this means sort the lines into increasing alphabetic order. The entire line is used as the sort key to determine this order.

Options For Sorting Files

- Lines are divided into fields
 - space/tab characters used as separator
 - allow more specific definition of sort keys
- Many other options

```
$ sort -nr -k4,5 userlist
sta4rf  tttyp3   Jan 23 (csun02)
roger    tttyp0   Jan 23 (csgi55)
james    tttyp4   Jan 23
dpm      tttyp8   Jan 23
dpm      console Jan 23
admis   tttyp7   Jan 23 (suna)
prj1gf  tttyp9   Jan 22 (blobby)
james   tttyp5   Jan 22 (csc$a)
csc6eq  tttyp2   Jan 22 (csgi47)
robert  tttyp6   Jan 21 (dosh$1)
csc5gf  tttyp1   Jan 19 (csgi05)
```

-n	sort on numeric value
-r	reverse the sort
-ka,b	sort key starts at field a (1-based) and ends at field b

sort offers many opportunities to control how the input lines are sorted. sort divides each line into fields (by default, separated by spaces or tabs), each of which can be used to construct the key used to determine the sort order.

sort provides options to change the field separator, to allow the sort to start at an arbitrary field (or sub-field). One important point to remember is that the field counter (and character counter within a field) are both 1-based, in other words the first field/character is indexed 1, rather than the Unix norm of 0.

sort also allows the main order to be reversed (from increasing to decreasing) and to use numeric comparison rather than alphabetic. The last is important since 19 is less than 2 according to the character values in the ASCII table, but clearly greater than 2 numerically.

Further options are available, consult the manual page for details.

Command Pipelines

- Display line 27 from a file

```
head -27 afile | tail -1
```

- Listing the ten largest files in the current directory

```
ls -l | sort -nr -k5 | head
```

- List only the sub-directories in the current directory

```
ls -l | grep '^d'
```

The philosophy of Unix is to:

write programs that do one thing and do it well

write programs to work together

write programs to handle character streams

The above examples demonstrate the utility of this philosophy. By using the pipe mechanism provided by the shell, the output of the left command is passed as the input to the one on its right.

The details of pipelining will be properly explained in a later module. It is enabled by using the pipe symbol | and by not supplying a filename to the right hand commands. When these commands realise that an input file has not been specified, they take their input from the standard input. The standard input is the keyboard if the program is executed directly, or the output of the preceding command if it is executed in a pipeline.

The first example shows the lateral thinking that must sometimes be applied in order to find an elegant solution. Having determined the first 27 lines, tail displays the last of these. Therefore, line 27 of afile is output.

In the next example ls generates a long listing of the files in the current directory. On Linux, the fifth field in the listing is the size of the file in bytes. The output of ls is sorted on the fifth field in reverse, numerical order. This is passed into head, which displays only the first ten lines of the input.

The last example uses grep to display only those entries generated by ls which begin with d. These are directories.

Comparing Files

- `diff` displays the differences between two files

```
$ cat story1  
once upon a time there were  
three bears that worked in  
the City  
  
$ cat story2  
once upon a time there were  
four bears that worked in  
the City
```

```
$ diff story1 story2  
2cl  
< three bears that worked in  
---  
> four bears that worked in  
  
$ diff -e story1 story2  
2c  
four bears that worked in  
.
```

`diff` displays only the differences between two files. Lines in the first file which differ from the second are prefixed with "<", lines in the second file which differ from the first are prefixed with ">".

A useful option for `diff` is `-e`. This causes `diff` to generate output which describes exactly how file2 can be generated from file1. The output is in the 'ed' editor language, and may be applied to `ed` automatically within a shell program (script). The technique is used by revision control systems to save having to store multiple copies of the same data on disk.

Other related commands include `cmp`, `comm` and `diff3`.

Cutting Fields

- **cut** removes selected fields from each line of the file
 - uses tab as field separator character
 - can also use character offsets to specify cut region

```
$ cat userlist
dpm      console Jan 23
roger    tttyp0   Jan 23 (csgi55)
csc5gf   tttyp1   Jan 19 (csgi05)
csc6eq   tttyp2   Jan 22 (csgi47)
sta4rf   tttyp3   Jan 23 (csun02)
```

```
$ cut -d" " -f1 userlist
dpm
roger
csc5gf
csc6eq
sta4rf

$ cut -c17-22 userlist
Jan 23
Jan 23
Jan 19
Jan 22
Jan 23
```

`cut` selects fields from lines or collections of characters from lines, excluding everything else. Effectively we are extracting one or more columns of data from the file.

In the above example, `cut` selects field one from file `userlist`, where fields are delimited by a single space. By default the field delimiter (separator) is a (single) tab. Beware, using a single character as the field separator as this can cause unexpected results; two of the characters together, will cause the second to be taken as the actual field.

Pasting File Contents

- Joins two or more files together
 - appends appropriate lines to each other
 - handles files of unequal length
 - use – to represent standard input as file to read from

```
$ cat file1
dave
pete
jane
paul
$ cat file2
1000
1001
1001
1002
$ paste file1 file2
dave    1000
pete    1001
jane    1002
paul
```

```
$ cat file2 | paste file1 -
dave    1000
pete    1001
jane    1002
paul
```

Paste allows us to combine the contents of two or more files, concatenating lines from the files into single lines in the output. If we reach the end of one of the files, then an empty string is inserted into the result.

We can also use the filename '-' to represent using standard input as one of the files to read from.

Duplicate Lines

- `uniq` removes duplicate adjacent lines from files

```
uniq -c      count number of duplicate line  
uniq -d      only print repeated lines  
uniq -u      only print non-repeated lines
```

```
$ cat userlist  
james  ttyp4  Jan 23  
robert ttyp6  Jan 21 (dosh$1)  
robert ttyp6  Jan 21 (dosh$1)  
admis  ttyp7  Jan 23 (suna)
```

```
$ uniq userlist  
james  ttyp4  Jan 23  
robert ttyp6  Jan 21 (dosh$1)  
admis  ttyp7  Jan 23 (suna)  
  
$ uniq -c userlist  
1 james ttyp4  Jan 23  
2 robert ttyp6 Jan 21 (dosh$1)  
1 admis ttyp7  Jan 23 (suna)
```

```
$ uniq -u userlist  
james  ttyp4  Jan 23  
admis  ttyp7  Jan 23 (suna)
```

`uniq` removes duplicate lines from files. Duplicate entries may appear when two or more files are merged (see `cat`) or when a file is updated by multiple processes. In order for `uniq` to work, the duplicate lines must be adjacent in the file. This is usually achieved by first sorting the file, and then by piping the output of `sort` into `uniq`.

```
sort filename | uniq
```

`uniq` does not have a `sort` operation built in since this contradicts the general philosophy in Unix of reuse. The shell provides glue which allows the output of one command to become the input of another, and thereby allows utilities to use the functionality of others.

Non-ASCII Files

- Octal dump `od` displays the contents of any file

```
$ cat story1
Once upon a time there were
three bears who lived in
the woods
```

```
$ od story1
0000000 067117 062543 072440 067560 020156 020141 064564 062555
0000020 072040 062550 062562 073440 071145 005145 064164 062562
0000040 020145 062542 071141 020163 064167 020157 064554 062566
0000060 020144 067151 072012 062550 073440 067557 071544 000012
0000077
```

`od` displays an octal, decimal, hexadecimal or ASCII dump of a file. It is useful when the file contains otherwise unprintable characters. Using `od` the characters are visible, displayed in a numeric or symbolic notation.

Non-ASCII Files

- Octal dump `od` displays the contents of any file

`od -x` display in hexadecimal
`od -c` display printable characters where possible
`od -d` display in decimal

```
$ cat story1
Once upon a time there were
three bears who lived in
the woods
```

```
$ od -x story1
0000000 6e4f 6563 7520 6f70 206e 2061 6974 656d
0000020 7420 6568 6572 7720 7265 0a65 6874 6572
0000040 2065 c562 7261 2072 c077 206f 696c 6576
```

```
$ od -c story1
0000000 O n c e   u p o n   a   t i m e
0000020   t h e r e   w e r e   \n   t h r e
0000040 e   b e a r s   w h o   l i v e
0000060 d   i n   \n   t h e   w o o d s   \n
0000077
```

In the example above, the space and newline characters of `story1` are visible. Note that lines in Unix files are separated by a single nl (newline) character, rather than the traditional carriage-return and line-feed characters. Also note that there is no end-of-file character.

Counting Words

- wc counts lines, words and characters

wc -l	just report lines
wc -w	just report words
wc -c	just report characters

```
$ cat userlist
dpm      console Jan 23
roger    tttyp0   Jan 23 (csgi55)
csc5gf   tttyp1   Jan 19 (csgi05)
csc6eq   tttyp2   Jan 22 (csgi47)
sta4rf   tttyp3   Jan 23 (csun02)
james    tttyp4   Jan 23
james    tttyp5   Jan 22 (csc$a)
robert   tttyp6   Jan 21 (dosh$1)
admis   tttyp7   Jan 23 (suna)
dpm     tttyp8   Jan 23
prj1gf   tttyp9   Jan 22 (blobby)
```

```
$ wc userlist
17   91   556  userlist
$ wc -l userlist
17   userlist
$ wc -w userlist
91   userlist
$ wc -c userlist
556  userlist
```

wc counts the number of words, characters and lines in a file. Any number of filenames may be specified and in the event of no filename, the standard input (output from preceding command or keyboard) is used.

Using command pipelines the number of files in a directory can easily be determined. If the output of ls is made the input of wc, then the number of lines is almost the number of files in the directory,

```
ls -l | wc -l
```

This turns out to be almost the right answer because ls takes the rather unusual behaviour of outputting a total in addition to one line per file. Consequently the count is out by one. The correct solution is determined by using

```
ls | wc -l
```

This works because ls outputs one filename per line whenever the output is redirected or piped. (ls cheats somewhat by only formatting its output if the actual stdout is a terminal device.)

wc may of course be applied to the output of any command to determine the number of lines, words or characters generated. To determine the number of users on the machine, for example,

```
who | wc -l
```

Printing Files

- `lpr` sends files to a line or laser printer

```
$ lpr -Plw story1
```

- `lpq` queries the state of the print queue

```
$ lpq -Plw
lw is ready and printing
Rank   Owner   Job     Files      Total size
active  dpm    555    userlist        385 bytes
1st    dpm    556    story1       456 bytes
```

- `lprm` dequeues jobs from the print queue

```
$ lprm -Plw 556
suna2: dfA556sparc2 dequeued
suna2: dfA556sparc2 dequeued
```

`lpr`, `lpq` and `lprm` send, query and delete print jobs. `lpr` copies each of the files specified to the printer queue, from which they will eventually be printed. `lpr` takes any number of filenames arguments, and reads standard input if none are given. The `-P` option identifies a different destination printer.

`lpq` lists the jobs waiting to be printed, including a print job number. The number must be used with `lprm` in order to remove the print job.

`lpr` may also be used at the end of a pipeline of commands, thus allowing the output of any command to be directly printed.

```
ls -l | lpr
```

Note that some versions of Unix/Linux provide alternative, though similar, commands for printing files, based around a command

```
lp
```

7 File Access Control

- File Access Rights 112
- Applying Access Rights 113
- Permission Bits 115
- Set ID Permission Bits 117
- Sticky Bit 118
- Setting Permissions 120
- Symbolic Notation 121
- Numeric Notation 122
- Default Permissions 123
- Changing Ownership 124
- Changing UIDs and GIDs 125
- Switching UIDs and GIDs 126
- Running Commands as Other Users 127

File Access Rights

- Controlling who has access to files and directories
 - preventing unauthorised access
 - preventing accidental damage
- Based on relationship between processes and files
 - every process is owned by someone and belongs to some group
 - every file is owned by someone and belongs to some group
- Two different approaches seen in Morgan Stanley
 - AFS access control used for all files/directories under /ms
 - "standard" Unix/Linux access control used for everything else



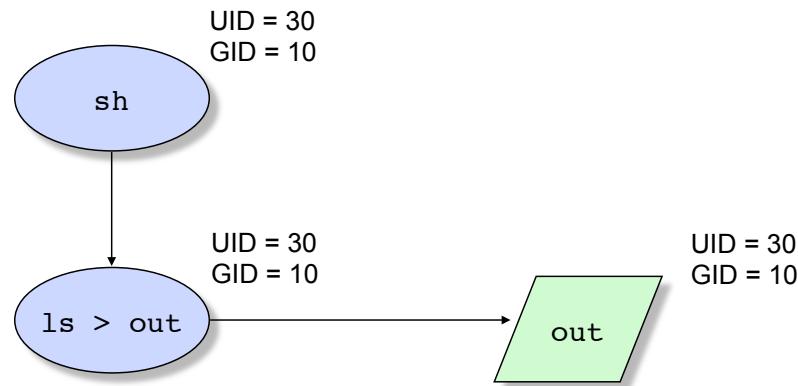
It is important that the resources (and in particular files) allocated to one user are not unexpectedly affected by another. Also, it is desirable to allow users to protect their own resources from accidental damage by themselves!

In Unix/Linux systems there has always been a simple but surprisingly effective mechanism for implementing access controls. AFS file systems introduced a different model that can potentially allow more fine grained control to be applied. However both models are based around similar ideas - every task performed by a user is implemented using a process, and every process is owned by some user (also belonging to a group). Every file system based resource (file, directory, device, program to run) is also owned by some user and belongs to a group. The relationships between these two ownerships can be specified and constitutes access control.

In this chapter we will first examine the standard Unix approach to access control - this will apply to all files except those located under the /ms directory. We will then look at AFS access controls.

Applying Access Rights

- When you login your shell is owned by you and is stamped with your UID and GID
- Almost every command you run and every file you create is stamped with your UID and GID



When a user first logs into the machine, they are requested to supply a username and password. The verification that the password is correct is the process of authentication. If successful a login shell is invoked on behalf of the user. The shell is stamped with the users personal UID and shared GID.

The UID (User Identity) is generally unique to the user, and has a one to one mapping with the user's username. The GID is shared amongst other users involved in the same activities. As shall be seen, users sharing the same GID may be able to access files which users outside of the group cannot.

Every process spawned by the user, inherits the UID and GID established in their shell. Every file created by the user is also marked with their UID and GID. In effect, everything that the user does during their session is marked with their identity.

Applying Access Rights

- Standard Unix model for access control is based on the relationship between processes and files



```
if UID (process) == 0
    allow access
else if UID (process) == UID (file)
    apply user's (owner's) access rights
else if GID (process) == GID (file)
    apply group access rights
else
    apply others access rights
```

Almost all activities on Unix involve the interaction of at least one process and one file. For example, to display a simple message the echo process would need to access the user's terminal file.

Access permissions are applied when processes access files. The standard Unix algorithm for applying access control is shown on the slide.

First, a check is made to see if the UID of the process is 0. This indicates that the process is being run by the "superuser". If the user is not superuser, then depending on the relationship between a process and a file, one of three sets of access permissions are applied. If both file and process are owned by the same person then the user (or owner) permissions are applied; otherwise, if both the file and process belong to the same group then the group permissions are applied; otherwise the other permissions are applied.

An important thing to notice with these permissions is that only one category can ever apply. It is possible that others may be able to access the file while the owner cannot! (However, the owner can change the permissions to grant themselves access.)

Super User

There are only two kinds of user in Unix, normal users and the superuser. The superuser normally has the username root, and unlike all other users is not bound by the permission mechanism.

Clearly the root user is dangerous. Users who find themselves requiring the privilege of root should take extreme care. Unix is unforgiving of errors.

Where network file systems are involved, a local root user does not normally have any special rights on remote files.

Permission Bits

- Every file carries permissions for its owner (user), group and others (everyone else)
- Permissions for files

r	<i>read</i>	read contents
w	<i>write</i>	alter contents
x	<i>execute</i>	attempt to run program
s	<i>set ID</i>	change the UID or GID of process

- Permissions for directories

r	<i>list</i>	list contents
w	<i>write</i>	create and delete <i>any</i> files
x	<i>search</i>	access the directory (with a <i>path</i> , <i>cd</i>)
t	<i>sticky</i>	finer control over write access to directories

Having determined the relationship between a process and the file it wishes to access, a set of permission bits are applied.

If the file is a plain file (non directory) then the process may be granted read, write, execute and/or set ID access. Read means the user may examine its contents (i.e., *cat* file), write means that the user may change the files contents (but not necessarily delete the file), and execute means that the user may attempt to run the file as a program. The file will crash if it does not contain program text. The set ID permission is only meaningful if the file contains a program, and causes the new process that executes this program to assume the identity (UID and/or GID) of the program file rather than the parent process, we will explain this in more detail shortly.

If the file is a directory, then the process may list the contents of the directory, create or delete files within the directory and/or search the directory. Note that x is overloaded to mean search not execute in this context (it does not mean anything to "execute" a directory!). Also note that the ability to write to a directory allows the process to overwrite or delete any of the files within the directory; even those for which the process does not have direct access. To modify this dangerous behaviour, the sticky bit on directories requires processes to have appropriate access to files within the directory, in addition to write access on the directory itself.

Permission Bits

- Use the `ls -l` command to display access rights

```
$ ls -l reports
drwxrwx--- 2 greg   dude      512 Jan 21 19:58 aDirectory
-rw-rw--- 1 greg   dude     873 Jan 21 19:59 aFile
```

type	user	group	others
d	rwx	rw-	r--
-	r--	r--	r--

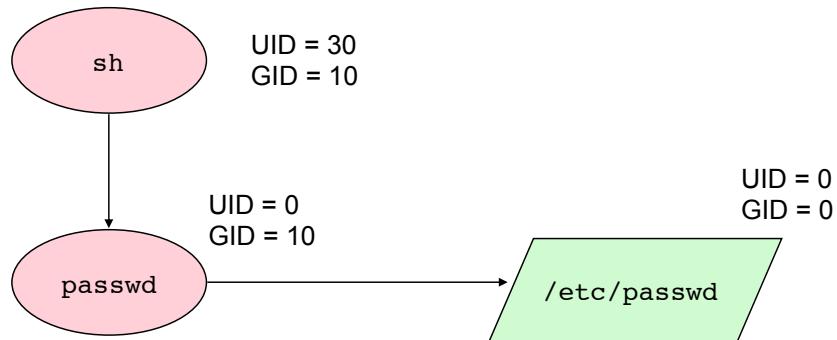
Use the `ls` command with the `-l` option to display the permission bits associated with a file.

The permission bits are organised in three sets of three bits, following the file type field. The sets identify the access rights for the owner (user), members of the group and all others.

Set ID Permission Bits

- Upon execution UID (& GID) are taken from program file
 - not from the parent process
 - for security, removed if file copied or written

```
$ ls -l /etc/passwd /usr/bin/passwd
-rw-r--r-- 1 root root 1852 2010-05-27 12:59 /etc/passwd
-rwsr-xr-x 1 root root 41292 2009-07-31 14:55 /usr/bin/passwd
```



In most situations, when a user invokes a new command the resultant process inherits the UID and GID from the parent. In this way, the UID and GID established when the user first logged in, is passed on to all child processes. Everything the user does therefore carries the user's identity.

This mechanism is sometimes insufficient. Consider the situation of a user wishing to change their password. Passwords are physically stored in an encrypted form within the /etc/passwd file. Although users are able to read the contents of the file, they do not have permission to write to it. If write permission was granted, then users would be able to add new users and set the passwords of others to null. This would entirely undermine Unix security.

The problem is that the passwd program needs to change the /etc/passwd file when a user updates their password. When the program is executed by a user it should inherit the UID and GID from the user's shell. However, if this was the case, the process would not gain write access to the passwd file. Only root has write access to this file.

The solution is the set ID bit. The long listing of the passwd command above (/usr/bin/passwd) shows an s in place of the user's execute bit. This is the set UID bit (in place of the group's execute bit it is called the set GID bit). The permission tells the system to run the passwd command with the UID of the owner of the command, not with the UID of the parent process. In effect, the passwd command runs as root, even though it is invoked by a normal user.

Sticky Bit

- A problem ...
 - write access on directory is all that is needed to delete a file
 - even if the file is owned by another user

```
$ ls -ld report
drwxrwxrwx 2 postgres george 4096 2010-07-12 11:34 report
$ ls -l report
total 4
-rw-r--r-- 1 postgres george 29 2010-07-12 11:34 secret
$ date >> report/secret
-bash: report/secret: Permission denied ←
$ whoami
george
$ rm report/secret
rm: remove write-protected regular file `report/secret'? y
$ ls -l report
total 0
$
```

No write permission to file

File deleted because we have write permission on directory

It is sometimes necessary for directories to be writable by many or sometimes all users. For example, /tmp is a general purpose directory which any process may use to store temporary files.

To achieve this, write permission is asserted for the user, group and other permission bits on the directory. A problem, however, is that write permission to a directory means that a user may create or delete any file in the directory, even if they do not have explicit access to the file itself.

In the example above, the file called secret, in the directory report, is owned by the user postgres. There is no write permission for other users to this file, so when another user (george) attempts to append content to the file the system rightly disallows this. However, although the file and the directory containing the file are owned by the user postgres, the other user is allowed to delete the file after confirming that the permissions are to be overridden.

Sticky Bit

- Problem resolved with the sticky bit
 - user has write permission on the directory, *and*
 - user owns the file, or owns the directory, or is superuser

```
$ ls -ld report
drwxrwxrwt 2 postgres george 4096 2010-07-12 11:48 report
$ ls -l report
total 4
-rw-r--r-- 1 postgres george 29 2010-07-12 11:48 secret
$ date >> report/secret
-bash: report/secret: Permission denied
$ whoami
george
$ rm report/secret
rm: remove write-protected regular file `report/secret'? y
rm: cannot remove `report/secret': Operation not permitted
$ ls -l report
total 4
-rw-r--r-- 1 postgres george 29 2010-07-12 11:48 secret
```

File not deleted

Here we see the same scenario, only this time the directory report has the sticky bit enabled. Notice how the rm command is not allowed this time.

The sticky bit allows a file to be deleted if

- the user has write permission on the directory, and
- the user owns the file, or owns the directory, or is the superuser.

Setting Permissions

- **chmod** is used to change permissions
 - only owner may change permissions
 - or superuser

```
chmod [-R] <permission-mode> file [files ...]
```

- The **-R** option enables recursion through a directory hierarchy
- The permission-mode may be specified using
 - a symbolic notation
 - a numeric notation

File permissions may be changed by the owner of the file or (of course) by the superuser.

The **chmod** command (change permission mode) allows the permissions to be changed on single files or directory structures. When specifying new permissions, either a symbolic or a numeric notation may be used.

Symbolic Notation

- The symbolic notation allows file permissions to be changed relative to the current permission

\$ ls -l aFile -rw----- 1 greg 873 Jan 21 19:59 aFile \$ chmod go+r aFile \$ ls -l aFile -rw-r--r-- 1 greg 873 Jan 21 19:59 aFile \$ chmod g-r aFile \$ ls -l aFile -rw----r-- 1 greg 873 Jan 21 19:59 aFile \$ chmod ug+rx,o-w aFile \$ ls -l aFile -rwxr-xr-- 1 greg 873 Jan 21 19:59 aFile	<p>chmod u + r files ... g - w o = x a s t</p>
---	--

In the symbolic notation codes are used to indicate who the permission should apply to, how the permission should be applied, and what permission is being changed.

The permission is applied to u (user), g (group) o (others), a (all of them), or any combination thereof. Permissions may be added, subtracted or assigned:

- + add new permission to the existing bits
- remove new permission from the existing bits
- = override existing permission with new permissions

The last example above demonstrates how several symbolic expressions have been combined using a comma.

Numeric Notation

- Treats the permissions as a bit pattern
- Each group of 3 bits is considered an octal value

s s t	r w x	r - x	r - -
4 2 1	4 2 1	4 2 1	4 2 1
0+0+0	4+2+1	4+0+1	4+0+0

= 754

```
$ ls -l aFile
-rw----- 1 greg dude 873 Jan 21 19:59 aFile
$ chmod 777 aFile
$ ls -l aFile
-rwxrwxrwx 1 greg dude 873 Jan 21 19:59 aFile
$ chmod 456 aFile
$ ls -l aFile
-r--r-xrw- 1 greg dude 873 Jan 21 19:59 aFile
```

The numeric notation is based on the idea that permission bits are either on or off, and that they therefore have a binary representation. Since the bits are clustered into three sets of three bits, a three digit octal number representing the permissions is easily determined.

Note that the numeric notation does not allow the permission bits to be changed relative to their current value. However, this is possible in the symbolic notation using the + and - operators.

An extra leading octal digit may be used with the numeric notation. This provides for set UID (4) set GID (2) and sticky(1). Setting a file's permissions to 7777 would enable everything, rwsrwsrwt.

Note, a lower case s or t implies that the x permission is also enabled. If x is absent, then the S and T appear as capital letters.

Default Permissions

- umask defines the default permission bits
 - defaults for files and directories created by process
 - not applied when files/directories are copied

```
$ umask  
22  
$ mkdir DIR  
$ touch FILE  
$ ls -ld DIR FILE  
drwxr-xr-x  2 greg  dude      512 ...  
-rw-r--r--  1 greg  dude          0 ...
```

r w x	r w x	r w x
4 2 1	4 2 1	4 2 1
0+0+0	0+2+0	0+2+0
4+2+1	4+0+1	4+0+1
4+2+0	4+0+0	4+0+0

022 umask

755 default for directories

644 default for files

__CODEDONE__

__CODE__CODEDONE__

__NOTES__When files and directories are first created, they receive a default set of permission bits. The default settings are based on a value maintained by the shell (all processes) called the umask. The umask masks out those permissions which should not be asserted by default. It is therefore the opposite to the permission actually given.

__NOTES__When a directory is created the maximum permission of 777 is masked by the umask. By default this is 022, giving rise to files/directories permissions (i.e., AND NOT umask with 777). In the case of plain files, it is inappropriate for them to have execute permission set by default (not many files are programs). Consequently, the starting permissions for files are 666.

Changing Ownership

- UID and GID of a file may be changed



- UID and GID of a process may be changed



If a process is unable to access a file, changing the files permission mode is not the only way to enable access. Permissions are based on the relationship of a file and a process. The relationship may be changed (and thus a different set of permissions will apply) by changing the UID or GID of the process or the file.

chown changes the ownership (UID) of the file, chgrp changes the group (GID) ownership of the file.

login causes the current user to logout and login as the new user, automatically attracting their UID and GID. su (switch user) allows one user to become another, leaving the original shell waiting. Once the user terminates the new shell, they return to the waiting parent shell.

newgrp allows the current shell to move into another group by changing its GID. To successfully execute this command you must be a member of the group you intend to move into (or provide a password if the group has been defined with one). The consequence of being in a new group is that access may now be granted to files based on the file's group permission bits.

Changing UIDs and GIDs

- Changing file ownership and group ownership

```
chown fred file
chown -R fred dir
chgrp staff file1 file2
chgrp -R staff dir1 dir2
chown fred:staff newfile
```

- Changing the current group

– you must be a member of the new group

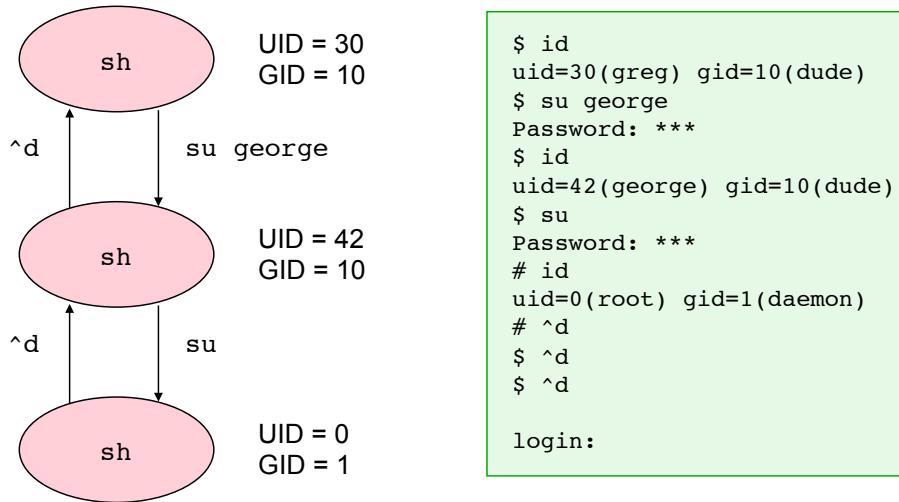
```
newgrp infotech
```

chown and chgrp require the new user or group, and the files which the command should be applied to. The operations may be applied recursively to a directory structure using the -R option.

In traditional Unix systems, users were permitted to change the ownership of files which they own. Therefore, one could give a file away but not take it back. However, due to various security problems associated with ownership, the chown command is no longer in the user's domain. Only the superuser can execute this command.

Switching UIDs and GIDs

- `su` enables one user to become another



`su` allows one user to switch to another. By providing the user's username and password, `su` creates a new shell with the new user's UID and GID. Any commands subsequently executed are done so with the identity of the new user.

In the example above, the `id` command is used to determine the current UID and GID of the shell. By using `su`, fred is able to become george, and then george is able to become root. As each of the shells are terminated, control returns to the waiting parent shell. fred is logged out when he terminates his login shell.

Note, if `su` is used without a username argument then root is the default. `su` is often used with a dash argument

The dash tells the new shell to configure itself appropriately for the new user, and to relocate itself in the user's home directory. Without the dash, little more than the UID and GID are changed.

In Kerberos Unix environments, use the command `ksu` to become another user rather than `su`, since this establishes the Kerberos tokens.

Running Commands as Other Users

- sudo command allows a single command to be executed as alternative user
 - root or specified user
- Sophisticated options based on config file
 - /etc/sudoers
 - defines per-user or group based permissions
- User must reauthenticate
 - as themselves
- Safer than su
 - no need to share passwords

```
$ whoami
george
$ cat /etc/sudoers
cat: /etc/sudoers: Permission denied
$ sudo cat /etc/sudoers
[sudo] password for george: *****
# /etc/sudoers
#
...
...
```

An alternative mechanism for allowing execution of privileged commands is provided by the sudo command, which allows a single command to be executed with specified user privileges (or superuser privileges).

The sudo command is preferable to su for a number of reasons. Details of who is allowed to do what are contained in a configuration file, /etc/sudoers, and can be specified to a fine grained level (much more than allowed by the standard Unix/Linux mechanisms). Users must authenticate to proceed with the sudo command, but they authenticate only as themselves, not as the user whose privileges they require. This removes the need to share passwords amongst users, especially superuser passwords.

8

Unix Editors

Unix Editors 129
ed Line Editor 130
Using ed 131
ed Commands 132
Appending Text into the Buffer 133
Inserting Text into the Buffer 134
Changing and Deleting Text 135
Reading and Writing Files 136
Text Transfer and Substitution 137
sed Stream Editor 138
Using sed 139
Text Substitution 140
Text Deletion 141
Printing Text 142
Reading and Writing Files 143
vi Screen Editor 144
Using vi 145
vi Modes 146
xemacs 147
xemacs Example 148

Unix Editors

- Editing text files a key facility on Unix/Linux
- Unix text editors exhibit different characteristics

ed	interactive, buffered, line oriented
sed	non-interactive, non-buffered, stream oriented
vi	interactive, buffered, screen oriented
vim	upgraded version of vi
xemacs	popular open source editor

Unix provides a variety of editors. Lots have been with the system for many years, others are recent editions. The three editors presented in this chapter are likely to be available on all versions of Unix.

ed is the simplest editor. It is interactive, buffered and line oriented. Files being edited are copied into memory, and edits are applied interactively to the copy. When complete, the changes may be written back (or discarded) to the original file. ed's line orientation means that lines are edited one at a time.

sed is a streamed version of ed. Rather than copying the file into memory, sed successively reads lines from the input file, edits them, and then writes the output to the standard output. sed is useful within pipelines as a programmable filter.

vi is the primary full screen editor on Unix systems. Like ed it works on a copy of the file in memory, but unlike ed it allows the user to view the file a page at a time. vi (pronounced vee-eye) is an extremely powerful and sophisticated editor. As such, the learning curve tends to be rather steep.

Editors have always been near the heart of the Unix cult. Users either love or hate them. Another popular editor, which is often used in place of vi, is emacs. However, since this is not shipped with the Unix operating system, it is not always available.

ed Line Editor

- ed is an basic text editor
 - line oriented operation – not visual in any way
 - the original text editor on Unix
- ed is available on all Unixes whatever their state
 - minimal systems always make ed available
- ed will work on any kind of terminal
 - does not requires special screen handling capabilities
- ed can be used in batch mode
 - it can be used inside shell scripts for automation
 - allows programmatic editing of huge number of files

ed has the distinction of being the oldest editor shipped with Unix. Although other editors are used in preference for day-to-day editing needs, ed is still useful in a number of situations.

When Unix systems are in their maintenance mode (in SunOS this is referred to as miniroot), the number of commands available is dramatically reduced. Since ed has the minimal system requirements (in terms of memory and terminal capabilities) it is often available when other editors are not.

ed is also useful within shell scripts. These are small programs containing sequences of shell and Unix commands. By embedding ed within such scripts, it is possible to automate the editing of a number of files. The editing could also be performed at night, if execution of the script is delayed until this time.

Using ed

```
ed [-p prompt] [filename]
```

```
$ ed bond
bond: No such file or directory
a
A debt investment in which an investor loans money to an
entity (corporate or governmental) that borrows the funds
for a defined period of time at a fixed interest rate.
Bonds are used by companies, municipalities, states and
U.S. and foreign governments to finance a variety of
projects and activities.
.
w
304
q
$
```

- Files are copied into memory for editing

ed is not the most friendly of programs. When it is invoked the only dialogue into which it enters is to inform the user of errors. It writes a ? whenever something is wrong.

ed does not even use a prompt by default. The -p is useful if a prompt is required when using the program.

When ed starts it reads the contents of the file being edited into memory, displays the number of bytes read, and then waits. If the file does not exist, then a message is displayed to that effect (?<file> on Solaris).

ed has two modes of operation. The command mode allows users to supply commands to move around the file, perform copies and substitutions, and to write the edits onto disk. Some commands move the user into input mode. In this mode everything which is typed goes directly into the file, until a . is typed at the beginning of a line.

In the above example the session starts off with ed indicating that the file does not exist. Currently, the session is in command mode. By typing the command a, the session moves to input mode. All the characters subsequently typed go into the buffer. When complete, the full-stop (period) . returns the session to the command mode. The buffer is then written to the file (by default, the one used when ed was started), and the session terminated.

ed Commands

- Commands have a simple and regular structure

```
[addr[,addr]]<character command> [parameters]

1           i           insert
5,20        a           append
/fred/      c           change
g/jane/     d           delete
/john/,/jack/ p           print
               w file    write to file
               r file    read from file
               s/RE/RS/g substitute RE for RS
               t addr   transfer to addr
               q         quit
               Q         really quit
```

ed commands conform to a regular structure. They are used in the command mode of the editor to perform edits on part or all of the file.

Some commands cause the session to change to input mode (i, a, c), others causes lines to be deleted or printed (d,p), and others cause the data to be written to or read from a file.

Each command applies to the current line unless an address is specified. The current line is initially the first line in the file, but this will change to the most recent line edited, as the session progresses. To set the current line, simply type in a line number.

By using an address, commands may be applied to specific lines or (where appropriate) to ranges of lines. A specific line is indicated by using its line number or supplying a regular expression (RE). In the latter, the editor will skip forward until a line containing the RE is found. A range of addresses can also be supplied, ranging positively through the file. Ranges are defined between two line numbers or two REs, and are used inclusively. As a convenience, '.' refers to the current line, and '\$' to the last.

Sometimes it is useful to search a file for all lines which contain an RE and apply an edit. This is achieved with the g qualifier to the RE address. For example, to print all lines containing a specified RE

`g/RE/p`

In fact, this activity was so useful that a grep command (g/re/p) was written as a general utility.

There are two quit commands. q quits only if all the changes have been written to a file; Q forces the quit and discards unwritten changes.

Appending Text into the Buffer

```
$ ed bond
304
$ a
Bonds are commonly referred to as fixed-income securities
and are one of the three main asset classes, along with
stocks and cash equivalents.
.
1,$p
A debt investment in which an investor loans money to an
entity (corporate or governmental) that borrows the funds
for a defined period of time at a fixed interest rate.
Bonds are used by companies, municipalities, states and
U.S. and foreign governments to finance a variety of
projects and activities.
Bonds are commonly referred to as fixed-income securities
and are one of the three main asset classes, along with
stocks and cash equivalents.
w
449
q
```

The bond file is edited once again and a new sentence is appended onto the end of the file. Note the use of the address \$, signifying the last line in the buffer and the command a, appending after the addressed line. Also note the use of . to switch from input mode back to command mode, so that the editor buffer can be written to disk.

Inserting Text into the Buffer

```
$ cat edtext
shaun  Shaun Fletcher p2
greg   G Fletcher     p4
greg   G Fletcher     p6
dpm    D Mallon      p7
neilb  Neil Bowers   p8
```

```
$ ed -p '>' edtext
100
>3i
john  J Jack          p5
.
>1,$p
shaun  Shaun Fletcher p2
greg   G Fletcher     p4
john  J Jack          p5
greg   G Fletcher     p6
dpm    D Mallon      p7
neilb  Neil Bowers   p8
>
```

In the above example, the text on the left is the original file, and the text on the right shows the edited file.

In this example ed is invoked on a file called edtext, and a > prompt is specified. Once started, ed informs the user that the input file contains 100 bytes.

3i inserts the following text before line 3. 1,\$p prints all lines from 1 to the end-of-file, thus verifying that the edit was successful. However, no changes have yet been written to the original file.

Changing and Deleting Text

```
$ cat edtext
shaun  Shaun Fletcher p2
greg   G Fletcher     p4
greg   G Fletcher     p6
dpm    D Mallon      p7
neilb  Neil Bowers   p8
```

```
$ ed -p '>' edtext
100
>/greg/c
john   J Jack         p5
.
>1,$p
shaun  Shaun Fletcher p2
john   J Jack         p5
greg   G Fletcher     p6
dpm    D Mallon      p7
neilb  Neil Bowers   p8
>

$ ed -p '>' edtext
100
>/Fletcher/,/Mallon/d
>1,$p
neilb  Neil Bowers   p8
>
```

The file is searched for the first line containing the RE greg. This is then changed (replaced) by the following text.

In the second example all lines between the REs Fletcher and Mallon, inclusive, are deleted. This is verified by 1,\$p. Note that the original file is still intact because the changes have not yet been written.

Reading and Writing Files

```
$ cat edtext
shaun  Shaun Fletcher p2
greg   G Fletcher    p4
greg   G Fletcher    p6
dpm    D Mallon     p7
neilb  Neil Bowers  p8
```

```
$ ed -p '>' edtext
100
>3,$w temp
>Q

$ cat temp
greg   G Fletcher    p6
dpm    D Mallon     p7
neilb  Neil Bowers  p8
```

```
$ cat text
*** nearly time for tea ***

$ ed -p '>' edtext
100
>/dpm/r text
>1,$p
shaun  Shaun Fletcher p2
greg   G Fletcher    p4
greg   G Fletcher    p6
dpm    D Mallon     p7
*** nearly time for tea ***
neilb  Neil Bowers  p8
>
```

The file is searched for a line containing the RE dpm. Once found the contents of text are read and appended after this line.

In the second example, lines 3 to end-of-file are written to a file called temp. The original file, edtext, is left unchanged.

Text Transfer and Substitution

```
$ ed -p '>' edtext
100
>2,3t$  
>1,$p
shaun  Shaun Fletcher p2
greg   G Fletcher      p4
greg   G Fletcher      p6
dpm    D Mallon        p7
neilb  Neil Bowers     p8
greg   G Fletcher      p4
greg   G Fletcher      p6
>
```

```
$ ed -p '>' edtext
100
>1,$s/G/Greg  
>1,$p
shaun  Shaun Fletcher p2
greg   Greg Fletcher   p4
greg   Greg Fletcher   p6
dpm    D Mallon        p7
neilb  Neil Bowers     p8
>
```

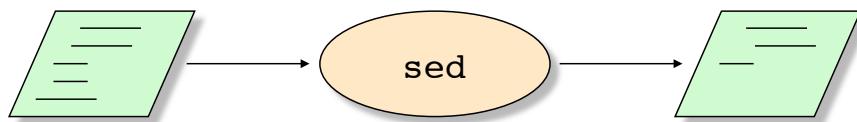
Note that both of the above boxes contain examples. In the left box all of the lines from 2 to 3 inclusive are transferred (copied) to the end of the file.

The right hand box shows the use of the substitute command. From line 1 to the end-of-file, the first G on each line is replaced with Greg. Note that further Gs on each line are left unaltered. To make the substitute command apply itself to all instances of G on each line, the g (global) qualifier must be used

`1,$s/G/Greg/g`

sed Stream Editor

- `sed` is a non interactive editor based on `ed`



- `sed` works on lines of text
- `sed` is a filter for text files

`sed` is similar to `ed` but works on streams of data. Rather than reading the entire file into memory, `sed` reads the input file line by line, performs the required edit, and then writes this to the standard output. The output either appears on the display or is passed through a pipe into another command.

`sed` is effectively a programmable filter. It allows the input stream to be filtered according to the edit being performed.

Since `sed` writes its output to the standard output, it does not change the file being edited. In fact, the usual source of input for `sed` is the output of another program.

Using sed

- Commands follow the same structure as for ed

```
sed [-n] '[addr[,addr]]<function>parameters' [files ...]
```

1	s/RE/RS/flags	substitute
5,20	d	delete
/fred/	p	print
/john/,/jack/	r filename	read file
	w filename	write file

- f allows a sed script file to be specified
- e precedes each edit when multiple edits are defined

sed commands may be specified on the command line or stored in a script file. The edits are described in an expression, rather than being interactively supplied from an environment. Consequently, the same edit expression is applied to each line of the input.

The edit expression follows the same format as that used by ed. The only exceptions, are that the /RE/ form of address applies to all lines containing the pattern, rather than simply the first; and without an address the edit applies to all lines in the file.

The other interesting thing to notice with sed, is the relationship of the delete and print commands. Since sed does not write changes back to the source file, the commands simply toggle as to whether or not a line should be printed. By default, sed prints all lines. Therefore, lines have to be deleted to stop them being printed. When the -n option is used, sed does not print lines by default. In this case, the only lines printed are those explicitly printed with p. Depending on the complexity of the expression, it is sometimes easier to delete lines which are not of interest, than it is to print those which are. Conversely, it is sometimes easier to print lines than delete them.

Text Substitution

```
$ cat sedtext
shaun  Shaun Fletcher p2
greg   G Fletcher     p4
greg   G Fletcher     p6
dpm    D Mallon       p7
neilb  Neil Bowers    p8
```

```
$ sed 's/ p/ Port-/' sedtext
shaun  Shaun Fletcher Port-2
greg   G Fletcher     Port-4
greg   G Fletcher     Port-6
dpm    D Mallon       Port-7
neilb  Neil Bowers    Port-8
```

```
$ sed '/Fletch/d' sedtext
dpm    D Mallon       p7
neilb  Neil Bowers    p8
```

```
$ sed -n '/Fletch/p' sedtext
shaun  Shaun Fletcher p2
greg   G Fletcher     p4
greg   G Fletcher     p6
```

```
$ sed -n '4s/D/Dave/p' sedtext
dpm    Dave Mallon     p7
```

In the following examples, the input file is on the left and the edits on the right.

In the first sed program no address is supplied so the command applies to all lines in the file. The command substitutes space followed by p, with Port-. Note that this selects only those lines with space followed by p. Also note that the space is used to identify the p at the end of each line and avoid other p's which may exist in the input file (for example, in dpm).

The next sed program deletes all lines which contain the RE Fletch. By default, sed prints all of the other lines. The opposite is achieved in the next example. In this, only lines which contain Fletch are displayed. This is because the -n option suppresses printing by default, and the p command causes lines matching the RE to be printed.

In the last example, lines are not printed by default, the edit is applied to line 4, and due to the p modifier on the substitute command, line 4 is printed after the substitution.

Text Deletion

```
$ cat sedtext
shaun  Shaun Fletcher p2
greg   G Fletcher     p4
greg   G Fletcher     p6
dpm    D Mallon      p7
neilb  Neil Bowers   p8
```

```
$ sed 4d sedtext
shaun  Shaun Fletcher p2
greg   G Fletcher     p4
greg   G Fletcher     p6
neilb  Neil Bowers   p8

$ sed '/^shaun/d' sedtext
greg   G Fletcher     p4
greg   G Fletcher     p6
dpm    D Mallon      p7
neilb  Neil Bowers   p8
```

In the first program line 4 is deleted. All other lines are displayed by default.

In the next example, all lines beginning with shaun are deleted, leaving the lines not matching this RE to be printed.

Printing Text

```
$ cat sedtext
shaun  Shaun Fletcher p2
greg   G Fletcher     p4
greg   G Fletcher     p6
dpm    D Mallon       p7
neilb  Neil Bowers    p8
```

```
$ sed -n '4,5p' sedtext
dpm    D Mallon       p7
neilb  Neil Bowers    p8

$ sed -n '/^g/,/^d/s/p[0-9]//p'
sedtext
greg   G Fletcher
greg   G Fletcher
dpm    D Mallon
```

Due to the `-n` option lines are no longer printed by default. However, lines 4 to 5 are printed explicitly.

The last example seems more complex than the others, but simply consists of an address and a command. The address is a range between two REs, the command is a substitution of p followed by any character, for nothing. Due to the `-n` option, the `p` qualifier (at the end of the line) has been added to the substitute command to ensure that lines involved in the edit are printed.

The sed program reads: from the first line that begins with a g until another line begins with a d (inclusive), substitute p followed by a digit for nothing. Do not print lines by default, only print lines involved in the substitution.

Reading and Writing Files

```
$ cat sedtext
shaun  Shaun Fletcher p2
greg   G Fletcher     p4
greg   G Fletcher     p6
dpm    D Mallon      p7
neilb  Neil Bowers   p8
```

```
$ sed -n '4,5w text' sedtext
$ cat text
dpm    D Mallon      p7
neilb  Neil Bowers   p8
```

```
$ cat text
*** nearly time for tea ***
$ sed '/p4/r text' sedtext
shaun  Shaun Fletcher p2
greg   G Fletcher     p4
*** nearly time for tea ***
greg   G Fletcher     p6
dpm    D Mallon      p7
neilb  Neil Bowers   p8
```

We use the `-n` option so as not to print lines by default. Lines in the range 4 through 5 are written to the file `text`.

In the second example, we search for all lines containing the regular expression `p4`, and insert into the edit buffer (after the line) the contents of the file `text`.

vi Screen Editor

- `vi` is an interactive editor
- `vi` is the most widely used Unix editor
- `vi` is based on `ex` (similar to `ed`)

Using vi

```
vi [-r] [files ...]
```

```
$ vi vitext
```

```
shaun  Shaun Fletcher p2
greg   G Fletcher     p4
greg   G Fletcher     p6
dpm    D Mallon      p7
neilb  Neil Bowers   p8
~
~
~
~
~
~
"vitext" 5 lines, 100 characters
```

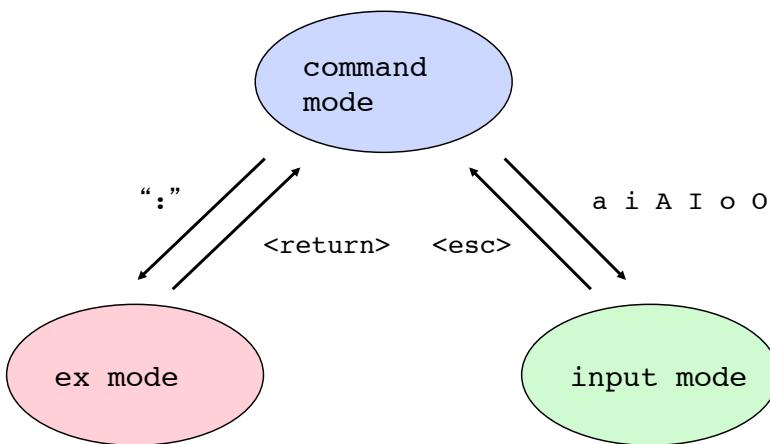
- Files are loaded into a buffer for editing

vi may optionally be invoked with a list of file names being those files to be edited. When many files are edited together, vi allows named paste buffers to be shared between the files.

The -r option allows files to be recovered in the event that the system crashes during an edit session. The mechanism works because vi maintains a journal file on the local machine from which the original file can be recovered. Note that using AFS and NFS users may login to virtually any machine to access their files. However, vi recovery journals are usually kept on the machine where the vi crashed. Therefore, login to the original machine to recover lost files.

vi Modes

- vi has three modes



In command mode virtually every character on the keyboard now defines a command. Some commands move around the document some command copy or delete text. Several commands change mode and move into the input mode. In this mode the characters have their literal meanings and go into the file. To leave input mode type escape.

The ex mode supports the ed and sed commands previously discussed. Use the ex mode to apply global operations, to insert other files and to write and exit the editor.

xemacs

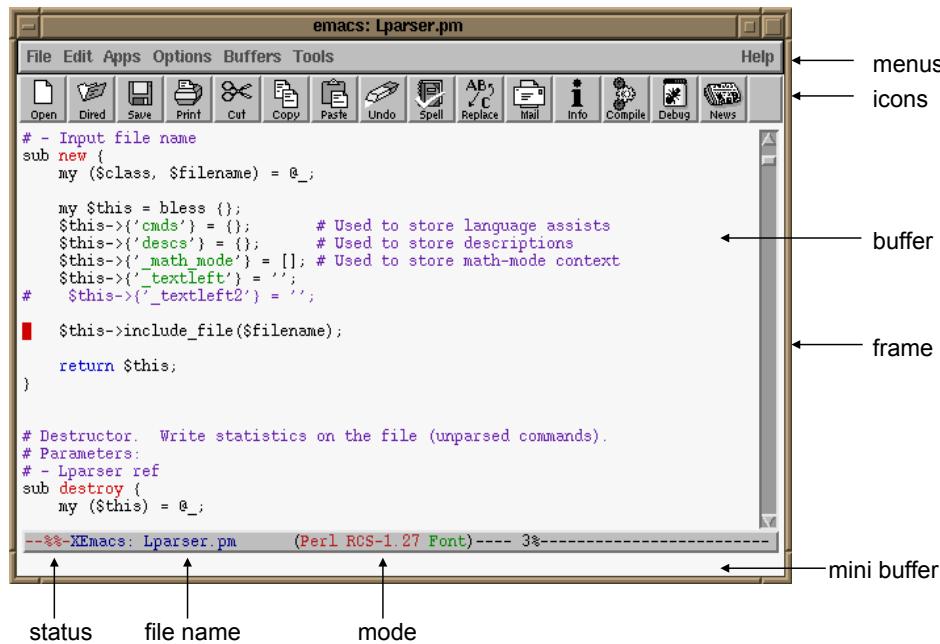
- emacs/xemacs is an extensible editor
 - written in C
 - uses LISP as its macro language
 - comes with 70 MB of LISP libraries
- One of the most popular editors
 - no vi-like modes
 - gives visual feedback while editing
 - supports many programming languages
- Violates the Unix philosophy
 - users ‘live’ within emacs
 - not opened and closed for each file
 - libraries implement functions, no sub-programs called
 - programs compiled and run from within emacs
 - supports mail, news, web browsing within emacs

Various versions of emacs are available, but all support the same basic editing commands and all work within text mode, but some offer special features such as X window support.

The most popular version is XEmacs, currently in version 19.16 (stable) and 20 (experimental non-western character support). However, the default version of XEmacs installed is 19.11. In order to run an up-to-date version, you must type:

```
module load fsf/xemacs/19.15
xemacs &
```

xemacs Example



The screenshot above shows XEmacs 19.15 editing a perl source file. In this screen, the whole edit buffer is used for the source file. You can also split the screen horizontally, in which case the frame contains two or more edit buffers. Each edit buffer can be a different file, though you can also edit different portions of the same file. If you need more flexibility than that, you can open additional frames.

If your emacs does not look like this, various things may have happened:

You may be running an older version. In the 'Help' menu at the right-hand side, you can verify the version. If you are running emacs version 19.11, then you've forgotten to type the module load command.

You may be running emacs in text mode, in which case it does not know that an X window environment is available. This typically happens if you telnet, rlogin or rsh to a remote system and omit to set the DISPLAY environment variable.

The terms used in emacs may be a bit different from what you've seen before. As they are used throughout the program, it's worth discussing them.

frame: the whole edit window, including menus and icons. Multiple frames can be open at a time.

buffer: an edit area for a file in memory. Multiple files can be open at any one time, with the user switching between files using the 'buffer' menu

minibuffer: a one-line command area at the bottom of the frame

status: a change indicator. % for a read-only file; ** for a changed file not yet saved; blank for an unchanged editable file.

mode: the edit-support library currently actively supports features like syntax highlighting, bracket matching, etc.

CODEDONE

9 Unix/Linux Shell

- What is a Shell? 150
- Linux Shells 151
- Purpose of Shells 152
- How the Shell Works 153
- Shell Directives 154
- I/O Redirection 155
 - Redirecting Errors 158
 - Appending Output 159
 - Command Pipelines 160
 - Building Commands 162
 - Valid & Invalid Redirections 163
 - Background Commands 164
 - Composing Commands 165
 - Grouping Commands 166

What is a Shell?

- A shell is any program that provides an interactive environment for users
- Shells are not built into Unix, they are just programs in the file system like `ls` or `cat`
- There are a collection of Standard Unix shells
 - bourne shell `sh` `$`
 - C shell `csh` `%`
 - korn shell `ksh` `$`
 - bash `bash` `$`
- Many more public domain shells available
 - `ash`, `zsh` ...

The bourne shell (`sh`), developed by Steve Bourne, is the original Unix shell and is used predominantly by system administrators and for shell programming.

The C shell (`csh`), California shell, was introduced by Berkeley and is more user friendly than the bourne shell.

The korn shell (`ksh`), developed by David Korn, is a development of the bourne shell. It offers facilities of command line editing and job control. This is the standard shell in Solaris.

`bash` is the born again shell, an open source version of the korn shell; `rks` is the reactive kernel shell (it predicts user behaviour) and `ash` is an adventure shell.

Linux Shells

- Bash – the Bourne Again Shell
 - GNU Project
 - POSIX compliant
 - compatible with Bourne Shell
 - largely compatible with Korn Shell
 - powerful command line editing
- ksh
 - public domain version of ksh
 - largely compatible with standard version



Linux systems are normally distributed with a variety of command line shells. However only a small number are used widely and these tend to be based on the shells that have evolved over the years in the main Unix systems.

Linux has a version of the Korn shell, although for licensing reasons it was not possible to include the actual original version. Nevertheless the version included, known as pdksh, is largely compatible and serves well.

Many users who were originally familiar with csh have moved on to the tcsh, an enhanced version that includes support for command line editing.

However, the most popular shell on Linux systems is bash - the Bourne Again Shell. This was developed as part of the GNU project and provides compatibility with both Bourne Shell (making the porting of shell scripts straightforward) and some features of csh, and even ksh. However it is bash's powerful tools for command line editing that have made it popular, especially with users who may not be used to the Unix command line interface.

For absolute compatibility, most Linux systems also include the Bourne Shell, sh - it still has uses executing scripts that are part of the start-up of the system.

Purpose of Shells

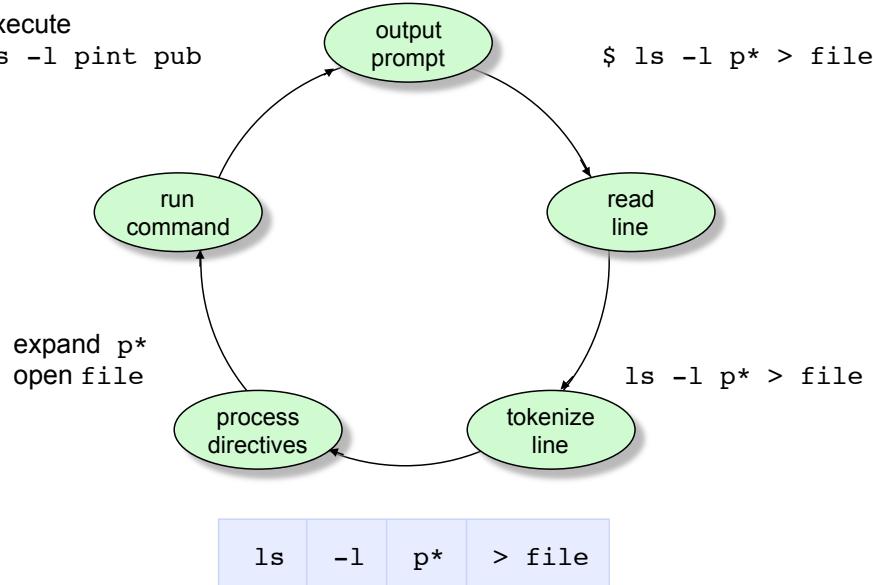
- Command line interpreter
 - reading commands from the user and executing them
- Programming language
 - processing a collection of commands in a file
- Job control language
 - enables commands to be executed at a later time
 - allows commands to be run in the background
 - allows commands to be chained together
 - enables command input/output to be redirected

The shell has three jobs; it is a command line interpreter, a programming language and a job control language.

As a command line interpreter, the shell prompts for input, reads whatever is typed, and then interprets this by invoking the requested command(s). Anything typed on the command line, may also be placed into an executable file called a shell script. This is useful if the same sequence of commands need to be executed many times, or needs to be invoked automatically. To support scripting, the shell also provides programming constructs such as while loops and if-then-else conditions.

The shell is also a job control language in that it helps users build-up and sequence commands, piping commands together, redirecting input and output streams, and generally controlling how a program is executed.

How the Shell Works



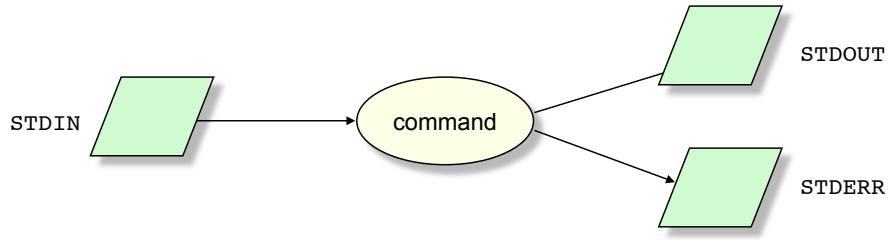
In addition to invoking commands, the shell can assist users by automatically generating file name argument lists. Using special wildcard symbols, the shell may be requested to complete a command by generating all file names which match the specified wildcard.

The shell also has considerable influence over the default input and output streams used by a child process. By default, processes read input from the keyboard and write output to the display. However, since these are simply files in Unix, the shell is able to change the files which these correspond to. In fact, the shell can even arrange that the output of one command is fed directly into the input of another.

Shell Directives

- Directives affect the way commands are invoked
 - redirecting command I/O
 - setting up command pipelines
 - running commands in sub-shells
 - conditionally composing commands
 - running commands in the background

I/O Redirection



- All commands expect three files to be opened for them when they are invoked

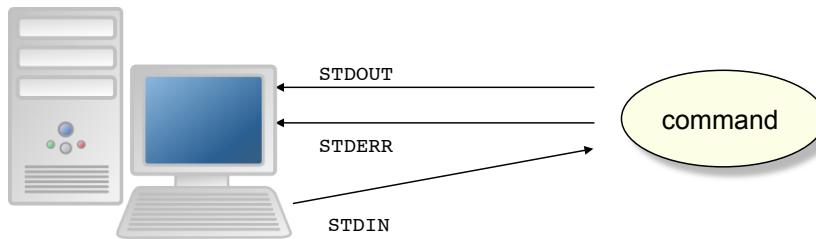
STDIN	standard input
STDOUT	standard output
STDERR	standard error

- The files are opened by the shell before it runs the command

The shell may be used to redirect input and output for commands. By default the input for a command (if not taken from a supplied file) is read from the keyboard (the standard input). Similarly, the output for a command is written to the display (or standard output) by default. The shell understands a number of special characters which direct it to change the files which constitute the standard input or output of a command.

I/O Redirection

- The shell binds STDIN, STDOUT and STDERR



- I/O directives change the bindings

	> output	redirect STDOUT to output
	>> output	append STDOUT to output
	< input	redirect STDIN from input
bash, ksh and sh	————→ 2> outerr 2>&1	redirect STDERR to outerr redirect STDERR to STDOUT's file
csh	————→ >& outerr	redirect STDOUT and STDERR to outerr

I/O redirection in Unix is based on the principle that all programs expect to be provided with three files from their calling process (usually a shell). The files are passed to the child process implicitly, by means of an internal file descriptor table.

When a command reads from STDIN (file descriptor 0) or writes to STDOUT (1) or STDERR (2), it is not aware where the bytes are coming from or being written to.

In stead, the process blindly reads and writes a file handle established by the parent.

The > indicates that the output should be written to the specified file, the < indicates that the input should be read from a specified file, and the | symbol indicates that the output of the left command should be directed to the input of the right command.

I/O Redirection

- Commands are unaware of the redirection

```
$ ls -l > longlist
$ cat longlist
total 3
drwx----- 2 fred      512 Jan 22 22:58 fileaccess
drwx----- 2 fred      512 Jan 22 15:11 filesystem
-rwx----- 1 fred      322 Jan 24 14:05 timetable
drwx----- 2 fred     1024 Jan 24 01:06 unixeditors
drwx----- 2 fred      512 Jan 23 21:19 unixfiles
-rwxr--r-- 1 fred      387 Jan 22 12:34 input

$ sort < input > output

$ mail greg < timetable
```

- Beware some redirections are dangerous

```
$ sort datafile > datafile
```

The danger in the above redirection for the sort command concerns the order in which the command is evaluated.

The shell reads the command first and handles the redirection. It opens the file datafile for output, implicitly overwriting the file by setting the file size to zero. The shell then invokes sort with the datafile as an argument. sort finds datafile empty, outputs nothing, and the file is left with zero bytes.

In general users should always be careful with redirections. In the case of sort, a -o option may be supplied to direct sort to write the sorted data back into the input file.

```
sort -o datafile datafile
```

Redirecting Errors

- STDERR may be redirected to a file

```
$ ls -l crud afile 2> err
-rwxr--r-- 1 fred  dude          34 Jan 13 22:34 afile
$ cat err
crud not found
```

- STDERR can be merged with stdout

```
$ ls -l crud afile > out 2>&1
$ cat out
-rwxr--r-- 1 fred  dude          34 Jan 13 22:34 afile
crud not found
```

The bourne or korn shell has been used in the above examples, since the C shell is rather limited in its control of STDERR.

In the first example STDERR is redirected to a file called err. Since STDOUT has been left untouched, the correct output of the ls command appears on the display.

In the next example, STDOUT is redirected to afile, and STDERR has been merged with it. Notice that the semantics of 2>&1 are to redirect STDERR to whatever file STDOUT is currently associated with. Since the expression is parsed from the left to the right, swapping the redirections around would stop the command working.

[2>&1 > out](#)

This causes STDERR to be mapped onto whatever file STDOUT is associated with (the display), and then STDOUT to be mapped onto out!

Appending Output

- The output of a program may be appended to the end of an existing file

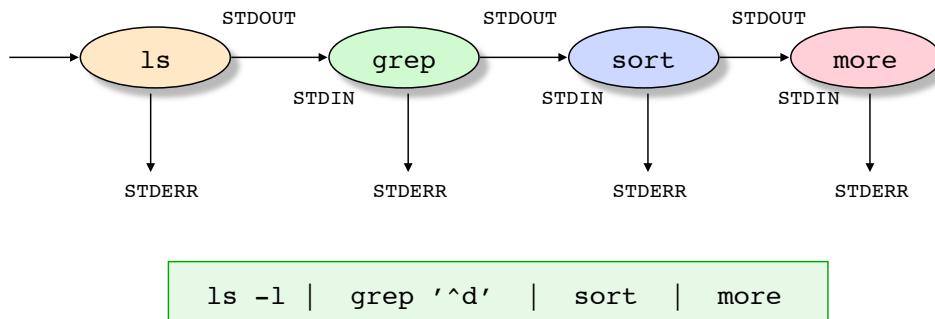
```
$ echo hello > output
$ ls -l afile >> output
$ cat output
hello
-rwxr--r-- 1 fred  dude          34 Jan 13 22:34 afile
```

- The reverse symbol << is used for ‘here’ documents
 - used in shell scripts
 - allow interactive input to be supplied within the script

Note that >> even works when multiple processes are simultaneously writing to the same file.

Command Pipelines

- The output of one command is directed into the input of another to form a pipeline



- By default, STDERR is not pass down the pipe

Standard error may also be passed through the pipe if this is required. In the C shell, the following syntax is provided

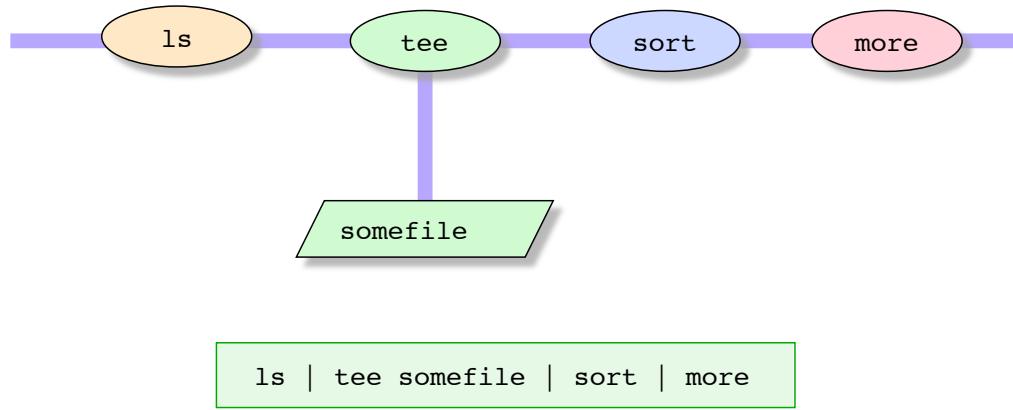
```
command |& command2
```

Both the normal output and the error output is passed into the pipe. In the bourne and korn shells, a different syntax is applied

```
command 2>&1 | command2
```

Command Pipelines

- tee enables the stream to be diverted



The tee command reads from STDIN and writes to STDOUT and to any files which are named as arguments. In the above example, tee reads the output of ls and writes this to somefile and passes the data to sort.

Duplicating the data stream in this way can give rise to sophisticated multi-pipeline techniques. Consider, a tail -f process in another window could be reading the text written to somefile and feeding this into another pipeline.

Building Commands

- Shells provide glue to build complex commands

```
$ who
dpm      console Jan 23 13:03
roger    tttyp0   Jan 23 14:10 (csgi55)
csc5gf   tttyp1   Jan 19 06:31 (csgi05)
csc6eq   tttyp2   Jan 22 21:11 (csgi47)
sta4rf   tttyp3   Jan 23 13:41 (csun02)
james    tttyp4   Jan 23 14:00
james    tttyp5   Jan 22 16:45 (cscSa)
robert   tttyp6   Jan 21 10:10 (doshS1)
admis   tttyp7   Jan 23 11:56 (suna)
```

```
$ who | cut -d" " -f1
dpm
roger
csc5gf
csc6eq
sta4rf
james
james
robert
admis
```

```
$ who | cut -d" " -f1 | paste - - -
dpm      roger    csc5gf
csc6eq   sta4rf   james
james    robert   admis
```

Unix provides a massive collection of utilities and filters, and the glue by which they can be combined to create powerful commands.

The output of who is fed into cut to remove all but the username, and then into paste to re-format the text into three columns. Note that the - symbols on the paste command line each represent the STDIN file. Therefore, as paste reads from each of the three files, it actually takes three more lines from STDIN.

Valid & Invalid Redirections

- Which of the following are valid?

```
process > file
file > process
file < file
process < process
process | process
process | file
file | process
file | file
process < file
```

Recall: files are passive and processes are active. A file cannot act upon another file or upon a process; however, a process may act upon a file or a process.

Background Commands

- Commands placed in the background do not cause the shell to wait

```
$ cc bigprog.c &
[1] 18581
$
```

- Programs running in the background should have their STDIN and STDOUT redirected

```
$ cc bigprog.c > diag 2>&1 &
```

A process is running in the foreground if the shell is waiting for it to complete---that is, if the process is in the foreground of the shell. In this case, the process' STDIN and STDOUT are usually tied to the terminal.

A process is in the background of its shell if the invoking shell does not wait for it to complete. Instead, both processes continue concurrently. In such cases, STDIN and STDOUT are usually redirected, since two processes can rarely sensibly read and write to the same terminal.

Composing Commands

- Sequential composition

```
com1 ; com2 ; com3 ; com4
```

- Parallel composition

```
com1 & com2 & com3 & com4
```

- Conjunctive composition

```
com1 && com2 && com3
```

- Disjunctive composition

```
com1 || com2 || com3
```

Commands separated by a semi-colon are executed in sequence; upon completion of com1, com2 is executed; when com2 is complete, com3 is executed. The same result may be achieved by placing the commands on different lines.

& is parallel composition, that is, all of the commands are executed at the same time. In the example above, com1 through com3 are executed in the background, com4 is executed in the foreground.

Conjunctive (AND) composition performs a logical AND between the return values of the commands, only continuing the sequence if the preceding command completes successfully. If com1 returns 0 (success), then com2 is executed, if this returns 0, then com3 is executed, and so on. The sequence stops when complete or if one of the commands returns false.

Disjunctive (OR) composition continues the sequence only if the preceding command fails. If com1 returns 0 then the sequence ends. If com1 returns non-zero then com2 is executed, if this returns 0 then com3 is executed. Conjunctive and disjunctive composition are useful in shell scripts as a short-hand mechanism for program control flow.

It is sometimes desirable to execute a series of commands sequentially and in the background. To achieve this use round brackets around the commands in the sequence. The shell will spawn a child shell in the background (and therefore not wait for it), and the child will run each of the commands in its foreground, waiting for each to complete and thereby controlling the sequence.

To redirect STDERR for an entire pipeline:

```
(ls | sort | more) 2>err
```

Grouping Commands

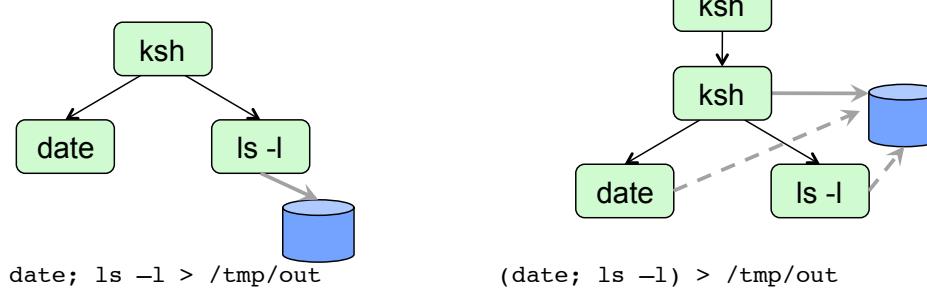
- Capturing output from a sequence of commands

```
$ (date; ls -l) > /tmp/out
```

- Running a sequence of commands in the background

```
$ (sleep 600; echo All Done) &
```

- Parentheses introduce sub-shell



Sometimes we need to group together a number of commands, usually running as a sequence, and treat them as if they were a single command. We would do this so that we could process all commands as one, for example to capture all the output from a sequence and do something with it (store it in a file or pass it through a pipe), or to use input redirection to a sequence of commands. Without the ability to group commands any I/O redirection would only apply to one of the commands in the sequence, usually the last. In the example on the slide, without the parentheses only the output from the `ls -l` command would be sent to the file `/tmp/out` with the output from `date` appearing on the default stdout for the command.

By introducing parentheses into the command, around the sequence, we cause the interactive shell to execute these commands in a sub-shell - in other words instead of directly invoking the two commands, the shell invokes a further instance of itself, whose I/O is redirected before the sub-shell invokes the commands. Since the commands get their I/O channels from their invoking shell, both commands will have their stdout redirected correctly.

A second use-case for sub-shells is where we wish to execute a sequence of commands in the background. Simply placing the '`&`' character after the last command in the sequence will cause the previous command(s) to execute in the foreground. Placing '`&`' after each command effectively gives us parallel composition. However using the parentheses adds a sub-shell, which runs in the background invoking the relevant commands one after the other.

Sub-shells allow sophisticated command sequences to be executed.

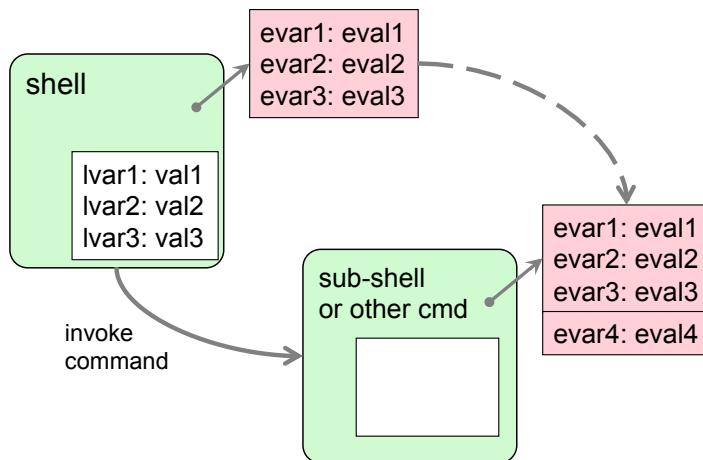
10

Shell Variables

Local and Environment Variables	168
Some Standard Variables	169
Examining Variables	170
Changing Variables	171
The PATH Variable	172
Morgan Stanley Module Commands	173
Module Commands	174
Quoting Mechanisms	176
Command Substitution	177
Interactive features of the Korn/Bash Shell	178
Command Line Editing	179
Interactive Usage of bash	181
Command Aliasing	182
Enhanced cd Command	183
Setting the Prompt	184
Command Line Editing with bash	185
Typical Unix Startup Files	186
Aurora Startup Files	187
Example Configuration	189

Local and Environment Variables

- Local variables are visible in the current shell only
- Environment variables are available in subshells and other commands



Local variables are part of the shell itself, and so are not visible to any commands (including sub-shells) that are invoked. However environment variables are held externally to the shell and passed to any commands or sub-shells that are invoked. This continues if the command or sub-shell itself invokes other commands.

Child commands are free to add new variables into the environment variables set, these are visible to any commands invoked from that point onwards, but not to the initial shell.

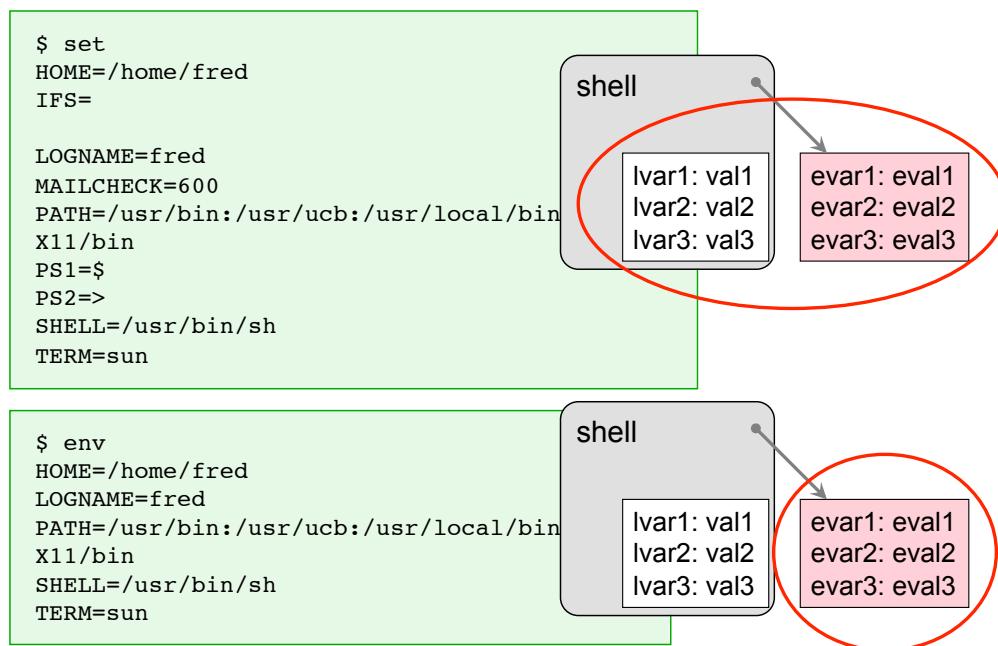
Some Standard Variables

- Pre-defined when the user logs in
 - they may exist in both locally and in the environment
 - their names depend on the shell being used
 - most can be modified by the user
- Some variables set by most shells

HOME	the path of the users home directory
LOGNAME	the users login name
PATH	the command search path for the shell
PS1	the shell's primary prompt
PS2	the shell's secondary prompt
SHELL	the path name of the shell
TERM	the type of the terminal
IFS	input field separator
MAILCHECK	the frequency e-mail is checked
DISPLAY	X server for GUI applications

In the C Shell, some different names are sometimes used. For example, TERM, SHELL, HOME and PATH are common to all shells; but in the C Shell PS1 is called prompt, and LOGNAME is called user.

Examining Variables



The shell command `set` will display all variables that are local and those in the environment. The command `env` will display only those variables in the environment. Notice that the `set` command has to be internal to the shell and not involve a separate process - otherwise it would not have access to the local shell variables, which are private to the shell... The `env` command is a standard command, which accesses only those variables that are in the environment outside the shell.

The C-shell uses a different mechanism for setting its variables and consequently environment variables do not start out as local variables. The commands to display csh variables are:

<code>set</code>	displays local variables
<code>setenv</code>	displays environment variables

Changing Variables

- Setting a local variable and exporting the variable

```
$ company=sun  
$ echo $company  
sun  
$ sh  
$ echo $company  
  
$ ^d  
$ echo $company  
sun
```

no spaces!!

```
$ company=sun  
$ echo $company  
sun  
$ export company  
$ sh  
$ echo $company  
sun
```

var=value	defines a local variable
export var	makes an environment variable
\$var	is replaced by the contents
export var=value	short-cut

It is important that there are no spaces around the = operator. If spaces are left, for example, after var, then the shell will attempt to execute var as a program.

In the C shell, variables are created and changed using the set and setenv commands

set one = unix defines a local variable

setenv two = vms defines an environment variable

The PATH Variable

- This is used by the shell to find commands
 - the shell searches each directory in sequence
 - the shell does not search the current directory by default
- Extending the PATH

```
$ echo $PATH  
/usr/bin:/usr/ucb:/usr/local/bin:/usr/X11/bin  
$ PATH=${PATH}:/home/fred/bin  
echo $PATH  
/usr/bin:/usr/ucb:/usr/local/bin:/usr/X11/bin:/home/fred/bin
```

prepend PATH /etc
append PATH /etc

ksh/bash short-cut
ksh/bash short-cut

- Placing . in the PATH should be avoided

A common practice is to place . as a directory in the PATH. This should be avoided, however, since it can sometimes lead to ambiguities. If it is necessary to execute a command in the current directory which is not in the PATH, then simply supply a path name

./program to run the command in current directory

Morgan Stanley Module Commands

- Allows the dynamic modification of a user's environment
 - installation and removal of software packages
 - based on modulefiles
 - focused on ksh/bash
- Ensures consistent addition or deletion of applications
 - sets / resets PATH, MANPATH, ...
- Syntax

```
module avail | list | display | load | unload package
```

The module command is used extensively in the Morgan Stanley Unix (Aurora) environment to effectively load and unload module packages from a user's environment. In practice all that the command does is to update key environment variables which enable the program to be found, and for its general settings to be managed.

avail - list available modules or sub-modules (if a module name is given)

list - lists modules that have already been loaded

display - shows what the equivalent module load would do to your environment

load - loads in a new module or sub module

unload - cleanly removes a module or sub module form your environment

Module Commands

- Which sub-modules are available for fsf ?

```
$ module avail fsf
----- /ms/dist/aurora/etc/modules -----
Versions of 'fsf' from '/ms/dist/aurora/etc/modules':
```

1.0	gcc/2.95.2.1	mconsole/prod
ImageMagick	gcc/2.95.3	nvi
ImageMagick/5.5.7	gcc/2.96	nvi/1.79
ast	gcc/2.96-112	nvi/beta
...		
flex	make/3.74.1	xemacs/21.1.14
flex/beta	make/3.79.1	xemacs/21.1.8
freetype	make/3.80	xemacs/21.4.10
freetype/2.0.9	make/3.80.1	xemacs/alpha
freetype/2.1.5	make/3.80.2	xemacs/beta
gcc	make/continuus	xemacs/release
gcc/2.8.1	mconsole	zephyr

Software packages in Morgan Stanley are organised with a well defined naming scheme that allocates a 3 part name to each. For example, the full name used for version 3.80 of make in the Aurora system is

fsf/make/3.80

We will see more about this organisation and naming scheme later.

Module Commands

- Which Modules are currently loaded?

```
$ module list
Currently Loaded Modulefiles:
  1) msde/prod          6) tex/prod
  2) fsf/1.0            7) fsf/xemacs/alpha
  3) 3rd/Purelink/1.2.1 8) 3rd/sunproc/6.2
  4) 3rd/sniff/prod     9) fsf/gmake/continuus
  5) 3rd/scopusqa/1.0
```

- Load a version of xemacs from module fsf

```
$ module load fsf/xemacs/21.4.10
$ module list
Currently Loaded Modulefiles:
  1) msde/prod          6) tex/prod
  2) fsf/1.0            7) fsf/xemacs/alpha
  3) 3rd/Purelink/1.2.1 8) 3rd/sunproc/6.2
  4) 3rd/sniff/prod     9) fsf/gmake/continuus
  5) 3rd/scopusqa/1.0   10) fsf/xemacs/21.4.10
```

Quoting Mechanisms

- Quoting mechanisms escape special characters

”...” escape wildcards and shell directives
’...’ escape \$ used in variable de-referencing
\ same as ’...’ for the following character

```
$ ls "p*"  
p* not found  
$ echo $home  
/home/sparc2/dpm  
$ echo "$home"  
/home/sparc2/dpm  
$ echo '$home'  
$home  
$ echo "it costs $10"  
it costs  
$ echo "it costs \$10"  
it costs $10
```

```
$ echo "hello  
> world"  
hello  
world  
$
```

The secondary prompt is used to indicate that the shell expects further input

Now we have seen that there are a number of special characters interpreted by the shell when processing an input command line - for example the I/O manipulation characters, filename wildcards, white space for separating arguments (newline to terminate the command) and now variable expansion.

There are, however, occasions when we do not want the shell to interpret these characters during input line processing. So we have a number of options for quoting characters in input. We have seen some of these already, but now the differences between them is more apparent. The main thing to remember is that within a section of the command line enclosed by double quotes "...", the special meaning of all characters is suppressed except for the \$ character, so that variables will still be replaced by their values. If the variable has no value then an empty string is inserted into the string. The single quotes '...' will suppress all characters. The backslash character suppresses any special meaning of the character next to it whatever that character is - even ' ' or \ itself.

Command Substitution

- The back quote `...` allows for command substitution
 - recent shells support `$(...)` notation
 - replace marked command with its *standard output*
- Shell metacharacters expanded in command
 - wildcards
 - variables
 - `$(...)` substitutions can be nested
- Useful for setting values of variables

```
$ date
Tue 13 Jul 2010 15:26:15 BST
$ date +%A
Tuesday
$ echo `date`
Tue 13 Jul 2010 15:26:34 BST
$ echo Today is $(date +%A)
Today is Tuesday
$ today=$(date +%A)
$ echo $today
Tuesday
$ echo `ls /tmp/ics*`
/tmp/ics17578 /tmp/ics24681
```

It is often very useful to build a command whose arguments are based on the results of another command. Consider how the rm command may be used to delete all files whose names are listed by ls. Using redirection is inappropriate:

```
ls | rm
```

This fails because rm is not expecting the list of files to be deleted to be supplied to its standard input. It expects the files to be provided as command line arguments:

```
rm `ls`
```

Of course, in practice one would simply write rm *, but the example nevertheless illustrates the potential of the back-quote mechanism.

Modern shells like ksh and bash support an alternative notation for command substitution:

```
rm $(ls)
```

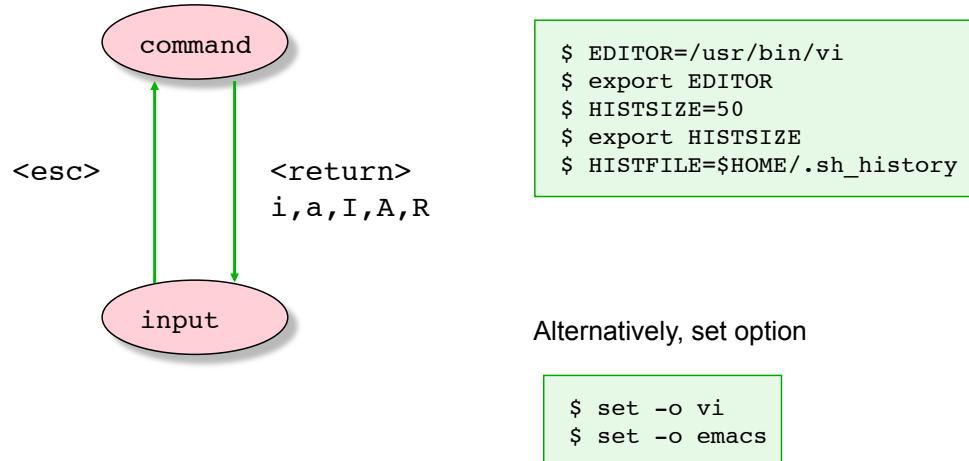
Note that in the command being run, shell metacharacters are expanded before execution of the command. Therefore it may contain wildcards, variable references, etc. The `$(...)` notation even supports nesting.

Interactive features of the Korn/Bash Shell

- Command Line Editing
 - re-run previous commands
 - edit and run previous commands
 - automatic filename completion
- Define Aliases
 - rename commands
 - define composite commands
- Additional Variables
 - sophisticated prompt
 - CWD, current working directory
 - ENV, define additional start-up file
 - HISTFILE, saves command line history

Command Line Editing

- Set up EDITOR and HISTFILE



Command line editing is not supported in the bourne shell. The C-shell provides a complex mechanism based on using an ! symbol. It is sufficient for re-running commands, but often too verbose for editing commands

```
set history = 50      enable history mechanism  
history   display command history  
!!       re-run last command  
!n       re-run command n  
!str     re-run last command beginning n  
^old^new^  substitute new for old in last command  
!n:s/old/new/  substitute new for old in command n
```

Command Line Editing

- A subset of the standard vi commands are available in command mode
 - enable with `set -o vi`
 - beware cursor arrow keys unlikely to work for movement

<code>k</code>	scroll back through command stack
<code>j</code>	scroll forward through command stack
<code>l</code>	move cursor right
<code>h</code>	move cursor left
<code>w</code>	move cursor right one word
<code>b</code>	move cursor left one word
<code>\$</code>	move to end of line
<code>0</code>	move to start of line
<code>x</code>	delete character
<code>dm</code>	delete text
<code>ym</code>	yank text
<code>p</code>	paste text

When the korn shell is placed into emacs mode (`set -o emacs`) the following standard emacs codes are enables:

```
<ctrl-p> previous line
<ctrl-n> next line
<ctrl-a> start of line
<ctrl-e> end of line
<ctrl-b> one character back
<ctrl-f> one character forward
<del> delete back char
<ctrl-d> delete next char
<ctrl-k> kill rest of line
```

Interactive Usage of bash

- Filename completion

– uses TAB key

TAB

```
$ ls /usr/incl  
...ls /usr/include/
```

- Use double TAB key for list of possible completions

TAB TAB

```
$ ls /usr/in
```

- Also works for commands

TAB

```
$ ema  
...emacs
```

Bash supports completion of filenames or pathname components by hitting the TAB key. If it is possible to complete the name, the command line is redisplayed with the completion having been done. If there is ambiguity, a bell is sounded to indicate that completion is not possible.

To find out the cause of the ambiguity, the user can hit TAB twice and a list of all the possible completions is displayed. The original line is displayed again and the user can add more characters as necessary to remove the ambiguity.

Filename completion as described above is available in a similar (though not so easy to use) way in ksh. However bash offers an even more convenient mechanism, in that it allows the same functionality to be used for command names, i.e. the first word in a command line.

Command Aliasing

- Allows symbolic names to be created
 - abbreviate long command lines
 - create new commands
 - BEWARE changing the behaviour of existing commands

```
alias [name['string']]
```

```
$ alias laser='lpr -Plw'  
$ alias dir='ls'  
$ alias r='fc -e - '  
$ alias mroe=more  
$ alias rm='rm -i'  
$ alias ls='ls -F'  
$ alias  
$ unalias rm
```

← requires the trailing space
← redefines rm
← lists aliases

The C-shell supports a similar style of command aliasing, except that a space is used in place of the = operator.

```
alias rm rm -i
```

Note that some aliases are DANGEROUS. In general, never change the semantics of an existing command. When you move to another machine (or user's account) the alias may not be defined!

It is a bad idea to alias Unix into your favourite operating system, i.e., aliasing ls to dir. The new command becomes a hybrid, being neither Unix nor the DOS. This makes life difficult when you move to another environment in which the aliases are not defined.

Enhanced cd Command

- Modern shells provide an enhanced cd command
 - supports the use of ~ substitution
 - allows toggling between directories
 - '-' character interpreted as "previous directory"

```
$ pwd  
/usr/X11/lib/fonts/100dpi  
$ cd ~fred/notes  
$ pwd  
$ /home/fred/notes  
$ cd -  
/usr/X11/lib/fonts/100dpi  
$ cd -  
/home/fred/notes  
$
```

- Tilde ~ may be used as a pathname specifier
 - without qualification it relates to the user's HOME directory

Tilde substitution also works in the C shell, but switching between directories does not. The C shell does, however, support directory stacking. Rather than cd to a new directory, the pushd moves to the directory but pushes the current directory onto a stack. Using popd the user can return to the previously pushed directory.

```
pushd /etc  
pushd /usr/bin  
popd  
popd
```

Setting the Prompt

- The prompt can be set to contain much useful information
 - can display details about the user and machine
 - can contain the current history number
 - may contain the current working directory

```
$ PS1=hello$  
helloPS1=  
PS1="$ "  
$ PS1="![!]$ "  
[33]$ echo hi  
hi  
[34]$ PS1='$PWD$ ' ← use single quotes  
/usr/X11/lib/fonts$ cd ..  
/usr/X11/lib$ PS1='![!]$PWD$ '  
[37]/usr/X11/lib$
```

use single quotes

- Take care not to make the prompt too long!

The C-shell allows the history number to be incorporated in the prompt:

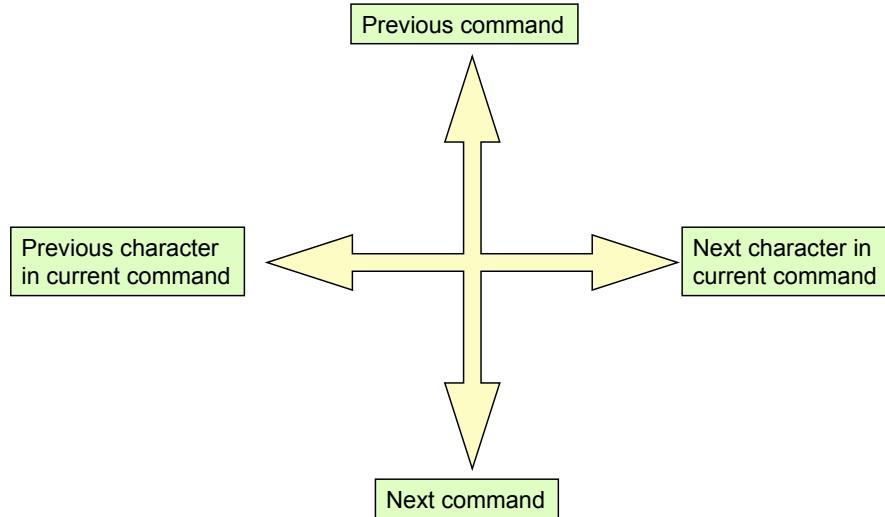
```
set prompt = "[!]% "
```

Storing the current working directory is more difficult. An alias has to be created for cd, such that each time the user changes their current working directory the prompt is also updated.

```
alias cd 'cd !* ; set prompt = "[!]$cwd%"'
```

Command Line Editing with bash

- Use cursor arrow keys on the keyboard



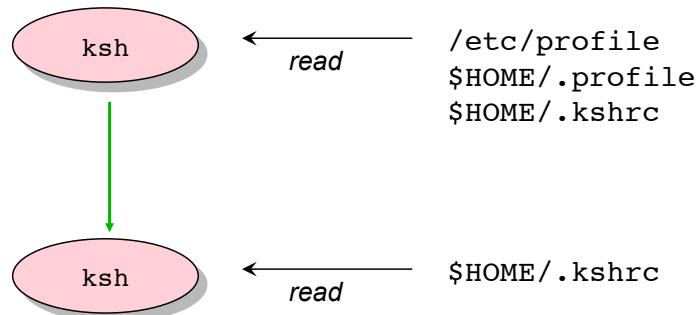
Perhaps the most convenient feature of bash is the way in which it supports command line editing. Rather than force users to remember edit navigation commands from vi or emacs, as ksh does, bash makes use of the cursor arrow keys on the keyboard.

There usage is as shown on the screen - to navigate through commands use the up and down keys. Within a command line use the left and right keys.

Within the command line, use backspace or the delete key to remove characters - any characters typed are automatically inserted into the command line. When the editing operation is complete, simply hit Enter and the command is passed to the shell - there is no need to position the cursor at the end of the line first.

Typical Unix Startup Files

- Used to configure the shell automatically
 - when the user logs in
 - when a new shell (perhaps in a window) is created



- The ENV environment variable must be configured

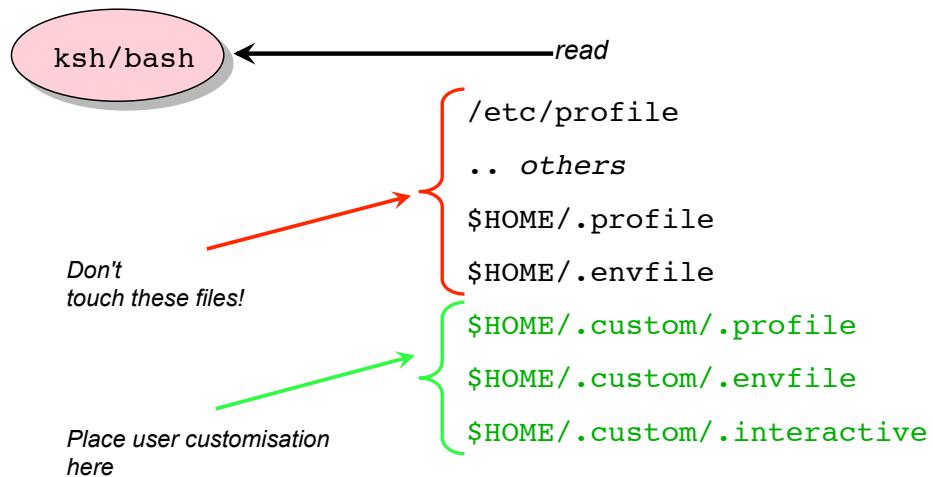
```
ENV=$HOME/.kshrc; export ENV
```

The Bourne shell reads from `/etc/profile` and `.profile` when the user first logs in. Nothing is read when subsequent shells are created.

The ksh shell is similar to the Bourne shell, except that it additionally reads from the file specified in ENV, when the user first logs in and with each subsequent shell. The ENV mechanism is useful for configuring child shells with information not passed through the environment variables.

Aurora Startup Files

- Chain of 16 or more startup scripts
 - mainly depending on user *model* (*programmer, trader, etc*)
 - model setup when user created, stored in `.model`



The Morgan Stanley environment has enhanced the structure the generic structure for account configuration somewhat.

Users should use configure their environment in `.custom/.profile`, `.custom/.envfile` and `.custom/.interactive`. (`.envfile` and `.interactive` are used in place of the conventional `.kshrc`)

User environments can be reconfigured using `installprofiles`, or from the USAG on-line tools.

Aurora Startup Files

- **/etc/profile**
 - sets the TERM variable
 - umask
 - issues message of the day
- **\$HOME/.profile**
 - Sets the ENV variable
 - `base.profile`
- **\$HOME/.envfile**
 - `base.envfile`
- **\$HOME/.custom**
 - directory for user-defined settings

The /etc/profile and the \$HOME/.profile files are read once when a user logs in.

The .envfile is re-read each time a user opens a new shell. The base files are located in the /common/etc/profiles directory and provide a standard set of MODEL user settings. The \$HOME/.profile file sets the ENV variable which defines which file to read when starting a new shell.

Each user may create their own .profile and .envfile in their own \$HOME/.custom directory. These are read in the same way as the system wide profile and envfiles.

Example Configuration

- **\$HOME/.custom/.profile**
 - defines the user's environment

```
PATH=${PATH}: ${HOME}/bin ; export PATH
PS1="[!]$ " ; export PS1

umask 077
PAGER=/usr/local/bin/less ; export PAGER
MANPATH=/usr/share/man:/usr/local/man
export MANPATH
LD_LIBRARY_PATH=/usr/X11/lib:/usr/lib
export LD_LIBRARY_PATH
HISTSIZE=50 ; export HISTSIZE
HISTFILE=${HOME}/.sh_history ; export HISTFILE
```

The .profile file contains mainly environment settings. Environment variables are placed here so that they can be passed into child processes.

Example Configuration

- **\$HOME/.custom/.envfile**
 - performs actions needed for each new shell
 - modifies the shell environment
 - also needed for non-interactive sessions

```
module load fsf/gmake
module load perl5/core/5.8
module load perl5-devel/5.8
module load fsf/gcc/default
module load msjava/sunjdk/1.5.0
set -o noclobber
```

The .envfile file also contains mainly environment settings. However, these are typically used on a per-shell basis.

The entries in this file will be used if you are starting a Korn shell, even if that shell is started by an automated process, such as Autostart or cron, where for example, modules are required to set the PATH variable for a tool that is called from inside a shell script.

You may also modify the behaviour of the shell here, to, for example, prevent clobbering of files.

Example Configuration

- **\$HOME/.custom/.interactive**
 - performs actions needed for each new shell
 - used for your interactive session
 - configures the interactive experience

```
EDITOR=vi #need to set here for vi mode cmd line editing
export EDITOR
alias gerp=grep
alias what=ls
```

The `.interactive` is used to modify the way that your interactive session behaves.

For example, if you prefer to use vi command line editing, then set the value of the `EDITOR` variable to `vi`. If you prefer using the Emacs editor command set, then set the value of the `EDITOR` variable to `emacs`.

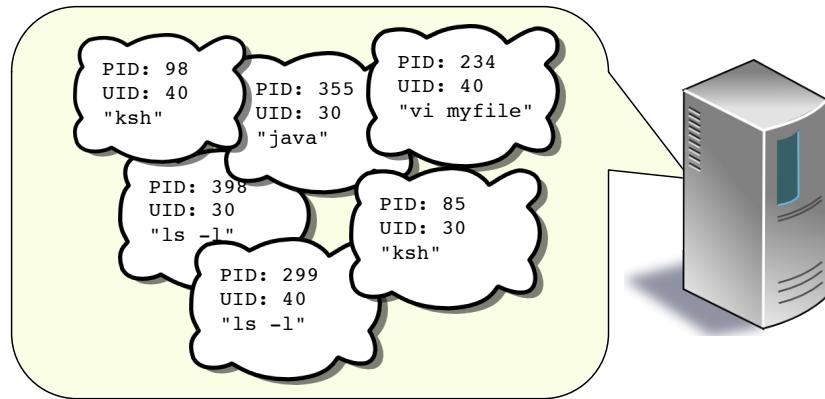
In the `.interactive` file, you can also define any aliases that you may require.

11 Controlling Processes

- About Processes 193
- Looking at Processes 194
- Lifetime of a Process 195
- Parent and Child Processes 196
- Processes and Jobs 197
- Foreground and Background Jobs 198
- Job Numbers 199
- Shell Job Control 200
- Example 201
- Signals 202
- Sending Signals 203
- Scheduling Repetitive Tasks 204
- Talking to cron 205
- Scheduling One-off Tasks 206
- Autosys 207

About Processes

- Each activity on a Unix/Linux system outside the kernel is performed in a process
 - a process is executing a program
 - a process has a unique numeric id
 - a process has an owner and a group
 - a process has a number of open files



As we have already seen, processes are a centrally important concept in a Unix system. Basically, a process is a particular instance of an executing program.

Every time any user executes a program (usually through issuing a command to the shell) a new process is created and acts as a container for that instance of the program. It is different from all other instances of programs in execution (even the same program), and identified by a unique numeric value called the Process ID or PID.

A process carries a number of attributes around with it, including its PID, details of the program that is running, ownership information in the form of a User ID and Group ID, and a set of open files and other system resources.

All activity outside the kernel (we call this "user level" activity) is carried out within processes. A primary job of the kernel is to coordinate all processes so that they have a fair share of the resources in the system.

Looking at Processes

- Use the ps command to examine processes
 - normally shows only locally interesting processes

```
$ ps
  PID TTY          TIME CMD
 1785 ttys000    0:00.15 -bash
```

- Different display options available
 - controlling what information is displayed...

```
$ ps -f
  UID  PID  PPID   C      STIME TTY          TIME CMD
 501  1785  1784    0  0:00.10 ttys000    0:00.15 -bash
```

– and which processes are displayed

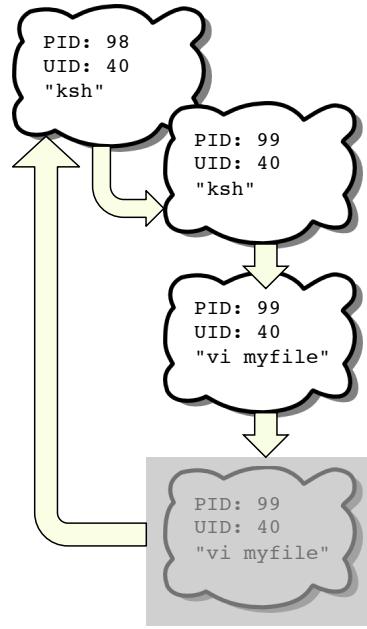
```
$ ps -ef
  UID  PID  PPID   C      STIME TTY          TIME CMD
 0     1     0     0  0:00.41 ??    0:00.55 /sbin/launchd
 0    17     1     0  0:00.65 ??    0:00.72 /usr/libexec/kextd
...
```

The ps command allows us to look at details of the processes that currently exist in the system. In its simplest form, the command shows basic information about "interesting" processes to the user. By "interesting", ps means processes immediately relevant to the current shell session.

There are a large number of options to ps, controlling what information is shown about processes, and which processes are displayed. Different versions of Unix may also have slightly different conventions for the information displayed by ps. You should always check the man page for ps to find out details.

Lifetime of a Process

- A process is created when an existing process is cloned
 - "fork"
 - new process is called the "child"
 - duplicate of its "parent"
- A process chooses to execute a program
 - "exec"
 - loads program from file and starts
- Process is destroyed when program terminates
 - "exit"
 - parent can "wait" for termination



The operations "fork", "exec", "exit" and "wait" are implemented as "system calls" - functions that are implemented inside the kernel. A Unix user has no direct access to these operations, however a C API exists so that programmers can build applications that make use of them. Indeed all the Unix/Linux shells have at their heart a loop that is continuously following this sequence of operations, and many network server applications such as web servers, mail servers etc. use this sequence extensively.

Parent and Child Processes

- When a new processes is created, it is an almost complete copy of the parent that created it
 - most attributes of the parent are passed to the child
- Some shared information
 - UID and GID
 - open files
 - umask
 - current program (code and data)
- Note that once the child exists
 - changes in the parent's attributes cannot affect the child
 - changes in the child's attributes cannot affect the parent

Clearly the child will have a different PID from its parent, but in most other respects it will originally be an exact copy. It will share the same ownership for permissions checking, the same open files, and the same program in execution.

Initially the code (text) and data will be the same but they are independent of each other so that once the new child process starts executing it will be updating its own copy of data and its own attributes.

Changes made in the child cannot affect the parent, and once created, changes in the parent cannot affect the child.

Processes and Jobs

- Most shell commands executed in a new process
 - except shell built-ins

\$ ls -l

PID: 299
UID: 40
"ls -l"

- Some commands result in multiple processes

- pipelines
- more complex applications

\$ ls -l | more

PID: 312
UID: 40
"ls -l"

PID: 310
UID: 40
"more"

Job %1

- Shell manages each command as a job

- allows multiple processes to be managed easily
- e.g. interrupted with Ctrl-C

As we have seen, the shell executes most commands apart from its built-ins in separate processes. For simple commands there will be one process for the command.

However, in the case of more complex commands, there may be more than one process. For example, in a pipeline there will be one process for each stage of the pipeline. Moreover, when we compile a C or C++ program, several processes are created, each to perform a different stage of the compilation.

It is convenient for the shell to be able to group together all processes concerned with a command and manage them as a single unit. This is known as a job. In fact every command that a shell runs is known as a job, and has a "Job Number" assigned to it.

Foreground and Background Jobs

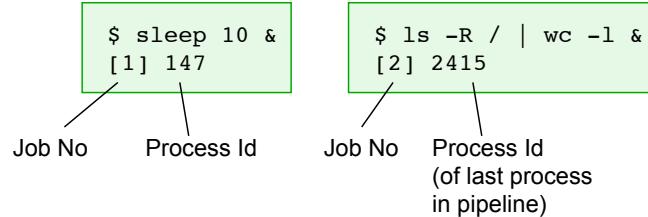
- A foreground job has access to the terminal for its input and output
 - shell waits for foreground job to finish before prompting for the next command
- A background job has no access to the terminal for input
 - may have access for output
 - shell will not wait for background job to complete before prompting for the next command
- Foreground and background jobs can be
 - suspended and resumed
 - terminated

We have already seen the definitions of foreground and background jobs, repeated here.

The shell allows a user to interact with executing jobs, foreground or background, to terminate them, or to suspend their execution and resume it later.

Job Numbers

- When background processes are invoked the shell tags them with a job number
 - displayed to the user



- Job No is not the same as process id
 - simple commands have one job no and one process id
 - process id of last process in pipeline shown

The Job number is a concept only known to the shell. Each process still has its own process id, and when a background job is run at the shell, both job number and process id is shown. When a pipeline command is run the process id of the last component of the pipeline is displayed.

Shell Job Control

- Allows the user to manage jobs
 - suspend/resume
 - set in foreground or background
 - terminate
- Uses combination of keyboard characters and builtin commands

<code>^Z</code>	suspends the foreground job
<code>^C</code>	terminates the foreground job
<code>fg</code>	resumes a suspended job in the foreground
<code>bg</code>	resumes a suspended job in the background
<code>jobs</code>	list currently active jobs
<code>stop</code>	suspends the specified background job (ksh)
<code>kill</code>	send signal to job (ksh and bash)

Modern shells allow jobs to be manipulated using a combination of keyboard control characters and builtin commands.

In this way it is perfectly possible to have many different tasks (jobs) underway at a time from a single shell session. Although a GUI makes this less critical than before, in practice much interaction with Unix is through a remote terminal interface such as ssh, where we only have a single shell with which to interact, so job control is still important.

Shell job control allows us to suspend a job that is running in either the foreground or the background. For a foreground job we use the Ctrl-Z character, for a background job the detail is slightly different for ksh than bash. ksh has the stop command, however with bash we have to use the slightly longer kill -stop command. Both of these take a job number as an argument:

`stop %1`

or

`kill -stop %1`

We can resume a stopped job either in the foreground (using the fg command) or in the background (using the bg command).

To terminate a foreground job we will typically use the Ctrl-C key. To terminate a background job we would use the kill command

`kill %1`

Example

- Some job control commands in action

```
$ sleep 1000 &
[1]      4524
$ ls -lR / > /dev/null 2>&1 &
[2]      452
$ jobs
[2] +  Running                      ls -lR / > /dev/null 2>&1 &
[1] -  Running
$ stop %2
[2] + Stopped (SIGSTOP)           ls -lR / > /dev/null 2>&1 &
$ fg %1
sleep 1000
^Z[1] + Stopped                  sleep 1000 &
$ bg %1
[1]      sleep 1000 &
$ jobs
[1] +  Running                      sleep 1000 &
[2] -  Stopped (SIGSTOP)           ls -lR / > /dev/null 2>&1 &
$ kill %1 %2
[1] + Terminated                 sleep 1000 &
[2] - Terminated                 ls -lR / > /dev/null 2>&1 &
```

Here we see some of the shell job control capabilities as presented in ksh.

With bash, the only difference is that instead of the

`stop %2`

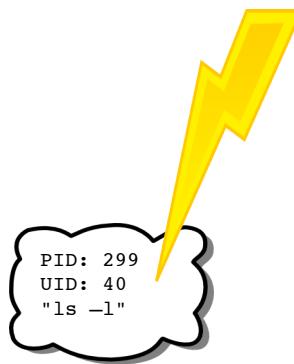
command, we would use

`kill -stop %2`

Signals

- Job control implemented using signals
- Signals are software interrupts delivered to processes
 - or jobs (groups of processes)
- Different signals identified by numbers
 - indicate different "events"
- Some common signals

2	Interrupt (Ctrl-C pressed)
15	Termination requested
11	Memory fault detected
17	Suspend requested
9	Hard kill



Behind the shell's job control functionality lies a more general capability of Unix to send messages to processes, indicating that some event has occurred.

These messages are known as signals, and they can be thought of as a software analogy to hardware interrupts.

There are a number of different types of signal that can be sent to processes, each indicating a different event that has occurred. The signal type is indicated by the "signal number". Current Unix/Linux systems support around 32 different types of signals, although some variants may have more than this. Most of the signal types are common to all variants, however.

Some signals are sent to the processes as a result of a condition detected by the kernel or hardware (for example a memory fault, or an arithmetic exception). Others are sent as a result of a request coming from a user or another process (for example a request to interrupt a process using Ctrl-C, or a request for the process to terminate). Each of the job control functions mentioned earlier is implemented by sending different signals to the appropriate process or processes.

One signal that needs special care is the "Hard Kill" signal (number 9). This signal is used to terminate a process that cannot be terminated in any other way, and will stop whatever the process is doing without giving a chance to clean up. Other signals allow a process to take some action before stopping if required. Sending signal 9 to a process should be considered a last resort.

Sending Signals

- Use kill to send a signal to processes or jobs

```
kill signal process_id  
kill signal %job_no
```

- Signal can be specified as number or as mnemonic

```
$ sleep 10000&  
[1]    4653  
$ ls -lR / > /dev/null 2>&1 &  
[2]    4654  
$ jobs  
[2] +  Running                      ls -lR / > /dev/null 2>&1 &  
[1] -  Running                      sleep 10000&  
$ kill -int %2  
[2] +  
$ kill -2 4653  
[1] +  
$ jobs  
$
```

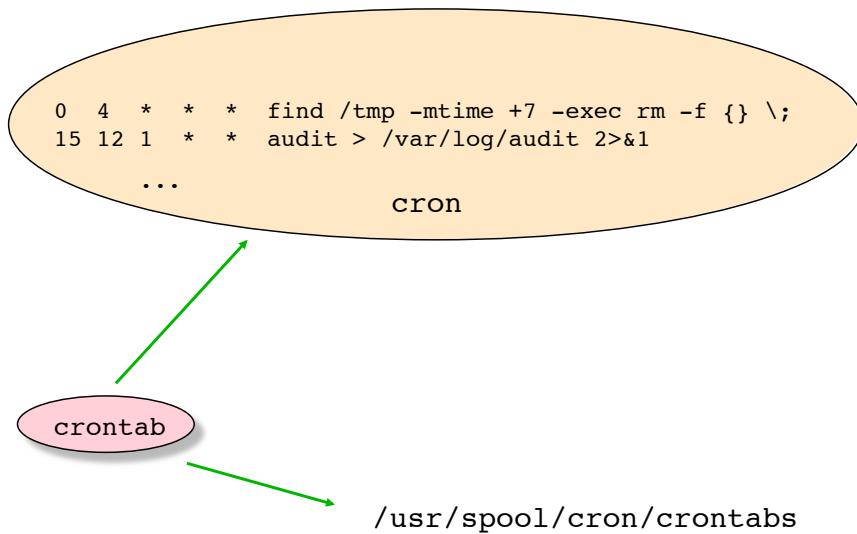
To send a signal to a process or a job use the kill command. The signal to be sent is specified either using its signal number, or by a mnemonic for the signal. So in the slide we see how to manually send the signal that is used to indicate that Ctrl-C has been pressed, using -2 for the number and -int as the mnemonic.

Notice also that we can signal a single process by using its process_id (which we could obtain from ps output) or its job number (which we must preface with the % character). If there are multiple processes associated with the job then all of them will have the signal sent to them.

PID	COMMAND
15596	-ksh (ksh)
15609	sleep 10000
15611	cc BigProg.c
15614	ps

Scheduling Repetitive Tasks

- cron runs commands at pre-defined times



cron is a system daemon which dispatches jobs (commands or scripts of commands) according to a given timetable. The timetable consists of entries, where each entry provides a time specification and a command to be run at the appropriate moment.

Users communicate their timetables to cron using the command crontab. This places the user into an edit session (by default vi) from which timetable entries may be specified. Since cron is a running process it will lose all timetable information when it is terminated. Consequently, crontab also stores timetable information in files within the spool directory. Note that each user will have their own crontab file within this directory structure.

The time specification allows repetitive events to be defined, so cron is often used by administrators to automate various system management tasks on repetitive schedules.

Talking to cron

- crontab sends a timetable of commands to cron

```
crontab [-elr] [filename] [username]
```

e	edit crontab entries
l	list crontab entries
r	remove contab entries

- crontab entries follow the same format



crontab is used to communicate a user's timetable of events to the cron process, and also to store the events for persistence. Using crontab users may edit, list and delete their cron entries. Only the root user can change another user's cron entries.

The format of a crontab entry is as follows. First, each line in the file represents a new entry. The first five fields specify the time, the last field the command. Minutes are 0 to 59, hours 0 to 23, month-day is 1 to 31, months 1 through 12, and weekday 0 through 6, where 0 represents Sunday. * may be used to specify all or don't care. Multiples and ranges are specified with , (comma) and - (dash).

```
0 0 * * * /usr/bin/clean
0 0 * * 1-5 /usr/bin/nightly
0 18 1 * * /usr/bin/monthly
```

To use an alternate editor with crontab, set the EDITOR environment variable.

Scheduling One-off Tasks

- One off tasks should be scheduled with at
 - front end to crontab

```
$ at 22:00
at> cc SomeBigJob.c
at> ^D
job 5397 at Sat Jan 30 22:00:00 1995
$
```

- Examining the at queue

```
$ atq
Rank  Execution Date          Owner      Job# Queue  Job Name
 1st   Jan 30th, 1995 22:00  fred       5397    a      stdin
$
```

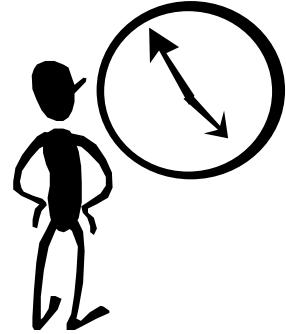
Unlike cron which is orientated towards repetitive tasks, at is useful for one-off tasks. The syntax is quite powerful and more intuitive than most Unix commands:

```
at now + 1 hour < someCommand
at 0930 Mar 10
echo "echo weekend" | at 5 pm Friday
```

at commands are managed by cron.

Autosys

- Production job scheduler
 - for distributed computing environments
 - Similar to cron, but boasts many more features
 - Used for production jobs only
- Advantages of Autosys
 - Correct handling if a host is unavailable
 - Easy visualization and management
 - Manages job dependencies
 - Calendars
 - Graphical and command line tools



Autosys is a job scheduling tool for Distributed Computing Environments (such as UNIX or Windows) that provides functionality and tools beyond the standard cron or at functionality available on those platforms.

It is similar to other distributed job schedulers such as Maestro or Control-M and Mainframe Schedulers such as CA-7/11 or Submitor.

12

Utilities and Filters

Working with Non-Text Files 209
File Compression 210
Text Formatting with nroff 211
Example - Manual Pages 212
Generating the Man Page 213
Spell Checking 214
Translating Characters 215
More Options for tr 216
Advanced Use of tr 217
Searching the File System 218
Searching for Commands 219
Searching for Files 220
Using the find Command 221
Search Criteria for find 222
find Examples 223
Manipulating Files 224
tar - Archiving Files and Directories 225
Using tar 226
Relative Path Names and tar 227

Working with Non-Text Files

- **strings** displays printable ASCII characters from a file

```
$ strings /bin/ls  
/lib64/ld-linux-x86-64.so.2  
...  
|$0H  
Try `%-s --help' for more information.  
Usage: %s [OPTION]... [FILE]...  
...  
Report bugs to <%s>.  
bug-coreutils@gnu.org  
LS_COLORS  
unrecognized prefix: %s  
target  
%*lu  
%lu  
cannot read symbolic link %s  
...
```

- Beware! Never hard-code passwords!

The command displays all bytes from a file that have values in the printable ASCII character range. This includes a number of meaningless sequences, but also string literals from the program, such as messages such as diagnostics but also, rather worryingly, passwords that have been hardcoded as strings.

File Compression

- **gzip** compacts the data in a file
 - gzip -9 selects maximum compression
- **gunzip** returns file to real size
- **zcat** (gzcat Solaris) uncompress to stdout
- **zgrep** grep through a compressed file

```
$ ls -l
-rw-r--r-- 1 george  george  201 14 Jul 12:41 input

$ gzip -9 input
$ ls -l
-rw-r--r-- 1 george  george  163 14 Jul 12:41 input.gz

$ gzcat input.gz
This is some input that I shall use to demonstrate the utility
of several ...

$ gunzip input.gz
$ ls -l
-rw-r--r-- 1 george  george  201 14 Jul 12:41 input
```

Unix supports a number of compression and decompression utilities. Perhaps the most widely used set manage compression into the well known zip format. gzip, gunzip, zgrep, gcat (gzcat on Solaris) are used for this purpose. Particularly useful is gcat, which decompresses a file onto the standard output, leaving the original compressed file intact on the file system. This saves having to recompress the file once the operation is complete. It also saves disk space, since there is no need to decompress the file in order to access the decompressed contents.

Note that compress will fail if it is unable to replace the file with a smaller version of itself. In the case, the original file remains intact with its original name.

The various compression utilities use the following filename extensions:

.z	pack, unpack
.Z	compress, uncompress
.gz	gzip, gunzip
.zip	zip, unzip
.bzip	bzip2, bunzip2

Text Formatting with nroff

- Unix was originally used for technical documentation preparation and management
 - based on [nt]roff text formatter

```
$ nroff input
This is some input which I shall use to demonstrate the utility
of several of the standard filters available on Unix. The file
is not too long but should be sufficient to show how the commands
operate.
```

- By default provides left & right justification
 - embedded markup commands allow much greater control
- Many packaged Domain Specific Markup libraries
 - man pages
 - mathematical equations
 - tables

roff is a text formatter as opposed to a text processor. This means that special codes are embedded within the source file to direct the formatting program when to centre, justify, start new paragraphs, etc.

Most document preparation is currently done on WYSIWYG window based text processors. However, text formatters are still very useful. For example, formatting commands can be generated by programs, creating text, graphs and tables.

Many of the traditional Unix books are written by die-hard roff enthusiasts. Another famous text formatter associated with Unix is TeX, and the associated macro language LaTeX.

In fact the techniques used within nroff are similar to those that underpin markup languages such as HTML and XML that are in wide use today.

Example - Manual Pages

- The manual pages are formatted with nroff

```
$ cd /usr/share/man/man1
$ gunzip -c ls.1.gz | more
.\" DO NOT MODIFY THIS FILE! It was generated
by help2man 1.35.
.TH LS "1" "October 2008" "ls 5.97" "User
Commands"
.SH NAME
ls \- list directory contents
.SH SYNOPSIS
.B ls
[ \fIOPTION\fR]... [ \fIFILE\fR]...
.SH DESCRIPTION
.\" Add any additional description here
.PP
List information about the FILES (the current
directory by default).
Sort entries alphabetically if none of \fB\-
cftuvSUX\fR nor \fB\-\-sort\fR.
.PP
```

The source files for the manual pages are stored in roff format. The embedded markup commands are usually introduced with a leading "." or leading "\". These are formatting instructions, for example, SH means format a sub-heading.

The manual pages exist in sub-directories from man1 to man8. Each of these directories contain a particular volume on the manual.

- man1 user commands
- man2 system services and error codes
- man3 library functions
- man4 device drivers and networks
- man5 file formats
- man6 games
- man7 document preparation
- man8 system administration procedures

Generating the Man Page

- Standard pipeline of Unix commands
 - man command

```
$ gunzip -c ls.1.gz | nroff -man | more
LS(1)                               User Commands
LS(1)

NAME
      ls - list directory contents

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

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

Here we see how the source of the manual page from overleaf is transformed into the page that is seen when we use the man command. The bulk of the formatting can be represented by the pipeline shown on the slide.

Notice the effect of the formatting of Sub-Headings - the .SH command causes the NAME, SYNOPSIS and DESCRIPTION lines to be shown with appropriate spacing and in a selected font. The detailed instructions on how to format sub-headings are stored in "macro libraries" that are specified as arguments to the nroff command - in this case the man page formatting instructions are stored in the "man" library.

The formatting shown here is relatively minimal as it is designed for a simple device (a character oriented terminal). However nroff and in particular troff are able to format their output from the same input files to a high level of accuracy to operate with sophisticated printing and typesetting devices.

Spell Checking

- spell displays words not found in the dictionary

```
$ spell input
available

$ look -b foot
foot
footage
football
footbridge
Foote
footfall
foothill
footman
footmen
footnote
footpad
footpath
footprint
footstep
...
...
```

spell -b uses British spellings

look searches the dictionary for
matching words

The dictionary is in

/usr/share/dict/words

grep through this to solve crosswords!

The spell command examines the /usr/share/dict/words dictionary file to ensure that all of the words specified on the standard input or in the input files, exist in the dictionary. Simple derivations of the dictionary words are also understood, such as plurals.

spell is not particularly intelligent, and the dictionary is quite small. However, spell does provide a useful checking tool which can be easily integrated into scripts and other programs.

A related command look can be used to check as to whether or not the specified word (or derived words) exist in the dictionary.

The dictionary itself may also be examined directly using grep. This is particularly useful when trying to solve cross-word puzzles!

An interactive version of spell, which will allow you to directly spell-check and improve a text, is available as ispell.

Translating Characters

- **tr** translates characters from the standard input
 - characters mentioned in first argument translated to corresponding character in second argument

```
$ tr '[a-z]' '[A-Z]'  
hello  
HELLO  
  
$ tr '[aeiou]' '[AEIOU]' < input  
ThIs Is sOmE InpUt thAt I shAll UsE tO dEmOstrAtE thE UtIty  
Of sEvErAl Of thE stAndArd filtErs AvAIlAbLE In UnIx. ThE fIle  
Is nOt vEry lOnG bUt shOUld bE sUFFIcIEnt tO shOw hOw thE  
cOmmAnds OpErAtE.  
  
$ tr ' ' '\n' < input  
This  
is  
some  
input  
that  
I
```

tr reads its standard input (if the data is in a file then the input can be redirected) and writes to its standard output. The first argument defines a string of characters to look for in the input, the second defines corresponding characters to be substituted in the output. Normally these arguments should contain the same number of characters. tr can translate to any character in the ASCII table by using the corresponding octal code. These are listed under man ASCII. In the above, \012 is new line.

When files contain unwanted characters use tr to translate them to a null or useful form. Look at the contents of the files using od, since this can display their octal, hex' or decimal value.

Note that tr only reads from standard input; there is no way to give it a filename to read from.

More Options for tr

- Delete characters from an input stream

```
$ tr -d '[aeiou]'  
Hello world  
Hll wrld
```

- Use complement of input character pattern

```
$ tr -cd '[aeiou]'  
Hello world  
eoo
```

- Character classes for commonly occurring character types

```
$ tr '[:lower:]' '[:upper:]'  
Hello, world  
HELLO, WORLD
```

There are many options to the tr command, which make it into a highly powerful and useful text processing tool. We have already seen the mapping of characters using equal sized maps. There are other options too.

We can ask that all characters in the "input" set are removed from the input stream when writing the output.

We can also use the -c flag to "complement" the match so that instead of matching characters in the string we match characters not in the string. In the second example on the slide we see that all non vowel characters are removed from the output. One thing to watch is that this includes the newline character at the end of the input line!

A number of special "character classes" are defined, which make the input and often output strings easier to specify. We see the examples [:lower:] and [:upper:], but there are many more such as [:alpha:] for alphabetic characters, [:alnum:] for alphanumeric characters, [:digit:] for digits and [:punct:] for punctuation characters.

Advanced Use of tr

- Different sized match and replacement strings
 - map all input characters to same output

```
$ tr 'aeiou' '-'  
The moon's a balloon  
Th- m--n's - b-ll--n
```

- map some input to one char, others to another

```
$ tr 'aeiou' '-+'  
The moon's a balloon, for sure it is  
Th+ m++n's - b-ll++n, f+r s+r+ t+ s
```

first char in input mapped to '-', others to '+'

- use -s (squeeze) option to remove duplicate replaced characters

```
$ tr -s 'aeiou' '--'  
The moon's a balloon  
Th- m-n's - b-ll-n
```

remove adjacent '-' in the output

For the basic usage of tr we would expect input and replacement strings to be the same length, so we have a one-to-one mapping. However they need not be.

If we have a single character in the replacement string, then all characters in the input string are mapped to that character.

If we have more than one character in the replacement string, then the first character in the input is mapped to the first in the replacement, the second to the second, and so on until the last character in the replacement string which acts as a replacement for all remaining characters in the input string.

When multiple characters map to the same output character, we can use the -s option to "squeeze" adjacent occurrences of any output character to one

Searching the File System

- Unix provides several commands for searching part or all of the file system
- Searching for commands
 - whereis looks in a pre-defined set of directories
 - which searches the PATH for the first instance
 - whence gives further information about commands (ksh only)
- Searching for files
 - find recursively searches any part of the file system



We may wish to search for commands to resolve issues related to running a particular version of a command, or to discover whether a command is implemented as a program or as a built in shell operation (the whence command is particularly useful for this although it is only available in ksh).

More likely, however, we would be searching to find a particular file in the filesystem. For this we would use the extremely powerful find command.

Searching for Commands

- **whereis** looks in the most likely directories
 - which searches user's PATH

```
$ which ls  
/bin/ls  
$ whereis vi  
/usr/bin/vi
```

- **whence** gives more information about the commands
 - ksh only

```
$ whence ls  
/bin/ls  
$ whence cd  
cd  
$ whence -v cd  
cd is a shell builtin  
$ whence -v whence  
whence is a shell builtin
```

whereis looks through a pre-defined list of directories (hard coded within whereis) to determine where the specified command should be. It is less commonly used today.

which follows the shell's PATH to see which instance of the specified command will be invoked if the command is given. It is particularly useful when there are several versions of a command available and there is ambiguity about which is being used.

ksh also provides the whence command, which can give more information about commands if used with the -v option. For example we see that it can help identify which commands are built in to the shell and which are implemented in other ways. Notice that the whence command is itself a ksh builtin!

Searching for Files

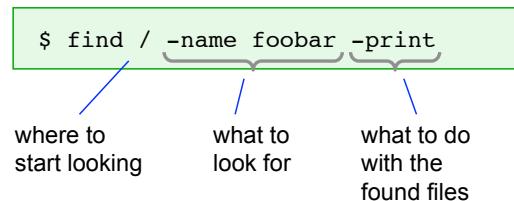
- **find** recursively searches any part of the file system
- Files selected based on their attributes
 - name, size, timestamps, ownership, etc...
- For each "found" file, various actions are possible
 - print the file name
 - apply any Unix command to the file

```
$ find / -name ls -print
/usr/bin/ls
/usr/lib/ls
/usr/sbin/ls
```

Find is an extremely powerful tool for searching through the filesystem hierarchy and applying operations to files and directories that have certain properties. It is one of the most common tools used by Unix/Linux system administrators, but can also be very helpful to general users.

Using the find Command

- Slightly unusual interface



- Each specified directory is root of search hierarchy
 - all files and directories visited
- Defaults supported
 - find everything
 - print out pathname of found items

The find command has one of the more unusual command interfaces in the Unix command set. It can often appear rather verbose and clumsy, but can be understood best as having three main parts:

- where the search is to start
- what search criteria to use
- what to do with each found file or directory

One or more start directories are specified, the command will recursively visit all of the contents of these.

find actually allows us to omit one or both of the "what to look for" and "what to do with it" parts of the command. Default behaviour is supported in each case
- by default we "find" everything and print out the pathname from the starting directory to the found file or directory.

Search Criteria for `find`

- "What to look for"
- `find` understands a variety of predicates for selecting files

```
-name <filename pattern>
-user <user name>
-group <group name>
-size [+|-] <size in blocks>
-perm [-] <octal number>
-atime [+|-] <days>
-mtime [+|-] <days>
-ctime [+|-] <days>
-type [d|f|l]
-inum <i-node number>
```

- For other possibilities check man page

The name predicate selects files which match the file name pattern. The pattern may be a single string, a wild card similar to the wild cards used by the shell.

Note, however, that since `find` has to interpret the wild cards, they must be quoted.

user selects files with owned by the specified user, and group selects files in the specified group. size selects files greater than (+), less than (-) or exactly equal to the specified size in 512 byte (0.5Kb) blocks. perm selects files based on their permission bits. Files exactly matching the specified octal pattern or files containing one of the specified permission bits (-) are selected.

atime, mtime and ctime relate to the files access, modification and inode change time stamps. The time is given as greater than (+), less than (-) or exactly equal to the specified days.

type is true for files of type d (directory), f (plain files) or l (symbolic links). inum is true if for files associated with the specified i-node number.

Other selection predicates may be supported as well as these common ones. Check the man page for `find` on your system for details.

find Examples

- List details of all files owned by root

```
find / -user root -ls
```

- Print pathnames of all files older than 3 months, which are larger than 100K

```
find / -atime +90 -size +200 -print
```

- Print pathnames of all C programs from the current directory

```
find . -name "*.c" -print
```

- Find all SUID root programs and e-mail administrator their pathnames

```
find . -user root -perm -4000 -print \  
| mail root@pluto
```

If find searches from a relative path name it generates relative names for the files which it finds. If it searches from an absolute path name then absolute names are generated.

This is relevant when find is being used to generate file names for archiving commands (cpio). Relative archives may be restored anywhere, whereas files in an absolute archives have to be restored into the same location from which they were copied.

Manipulating Files

- `find` can execute any Unix command on the selected files

```
find . -name core -exec rm {} \; -print
```

- All files named `core` files underneath the current directory are deleted
 - `{}` represents the file being processed
 - `;` terminates the Unix command (mandatory)
 - `\` escapes the `;` from the shell
- An alternative predicate prompts for confirmation

```
find . -name core -ok rm {} \; -print
```

The `exec` predicate makes any Unix command programmatically recursive.

tar - Archiving Files and Directories

- **tar** is a popular archiving tool for Unix
 - combines directory hierarchy into a single file
 - file can be a device, making tar a potential backup tool

```
tar [cxtvf] tarfile file [files ...]  
  
c      create a tar backup  
x      extract a tar backup  
t      list contents of tar backup  
v      give verbose information  
f      backup device file
```

- Archive file can be compressed
 - using tools described earlier
 - common mechanism for distributing Unix/Linux software

The tar command started its life as a tool for backing up files and directories to tape drive (the name means tape archiver). However in more recent times it is used as a convenient way of packaging directory hierarchies into single files that can then be copied around the filesystem or between machines.

Tar files can be compressed using the standard Unix/Linux compression tools, and this has become a very common way of distributing software such as applications amongst machines.

Using tar

- Creating an archive

```
$ tar cvf mystuff.tar Reports aFile
```

- Listing the contents of the backup

```
$ tar tvf mystuff.tar
```

- Extracting a file or the entire backup

```
$ tar xvf mystuff.tar afile  
$ tar xvf mystuff.tar
```

The command line syntax is again slightly strange; the arguments are one "major" instruction (for example c to create an archive, x to extract from an archive) and a number of modifiers to the command.

For example the 'v' option selects verbose operation - details of each file and directory being processed are shown on standard output. The 'f' option defines the pathname to the archive file.

Other options are possible.

Relative Path Names and tar

- Always use relative path names
 - when specifying files and directories to archive
 - gives greater flexibility when extracting

```
$ tar cvf archive.tar /home/user1/Reports
a /home/user1/Reports/cable1
a /home/user1/Reports/text
a /home/user1/Reports/style
...
$ tar xvf archive.tar
x /home/user1/Reports/cable1
x /home/user1/Reports/text
x /home/user1/Reports/style
...
```

One important point to remember when creating an archive is to avoid using absolute pathnames for the directories or files being archived. This will allow them to be extracted into a different directory than their original.

For example if we used the command

```
tar cvf mystuff.tar /home/user1/Reports
```

to create the archive, then when we extracted the file it could only be placed as /home/user1/Reports. However if we used

```
tar cvf mystuff.tar Reports
```

Then the file would be extracted into the current directory from which we executed the extract command. This gives much more flexibility.

13 Windows Basic Architecture

- What is Windows? 229
- History of 32-bit Windows Desktops 230
- History of Windows Servers 232
- Windows Architecture 233
- Windows Memory Model 234
- Devices and Drivers 235
- Device Manager 236
- Driver Signing 237
- Introduction to the Registry 238
- Registry Structure 239
- Editing the Registry 241
- Some Registry Tips and Tricks 242
- Default Windows Services 244
- Windows at Morgan Stanley 245
- Currently Deployed Windows Versions 246

What is Windows?

- 32-bit (or 64-bit) Operating System
 - multi-user, multi-process, multitasking OS
 - desktop and server versions
 - demand-paged, virtual memory system
 - apps run in their own, linear memory space
 - malfunctioning apps do not affect others or the OS
 - includes tools for administration and management
 - includes several services and applications



Windows is available as a 32-bit or 64-bit operating system.

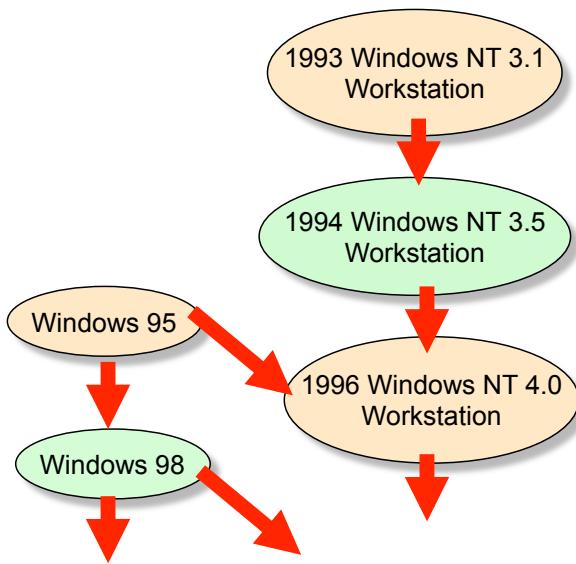
Windows provides multitasking and multithreaded operations. Different applications can run at the same time. Background applications can continue while a user works in the foreground. Multiple threads in an application can operate simultaneously.

Windows operating systems are optimized to work in two major environments; the desktop and the server. The desktop versions, such as Windows XP are tuned to provide a multitasking desktop operating system. The server versions are tuned differently and include additional features to provide an operating system for server-based applications.

The Windows virtual memory structure provides running applications some degree of separation from each other. Windows supports applications in separate memory address spaces. Applications running in separate memory spaces are protected from being affected by any malfunctioning applications.

Windows desktop operating systems include many productivity applications and the server operating systems include several services and applications. This means that the out of box experience (OOBE) provides enough capabilities to build, administer, monitor, and use a functioning network.

History of 32-bit Windows Desktops



Windows 32-bit desktop history.

1993: Windows NT 3.1

"Windows NT represents nothing less than a fundamental change in the way that companies can address their business computing requirements," Microsoft Chairman Bill Gates said at its release.

1994: Windows NT Workstation 3.5

The Windows NT Workstation 3.5 offered 32-bit performance improvements and better application support.

1995: Windows 95

Windows 95 was the successor to the three existing general-purpose desktop operating systems from Microsoft—Windows 3.1, Windows for Workgroups, and MS-DOS.

1996: Windows NT Workstation 4.0

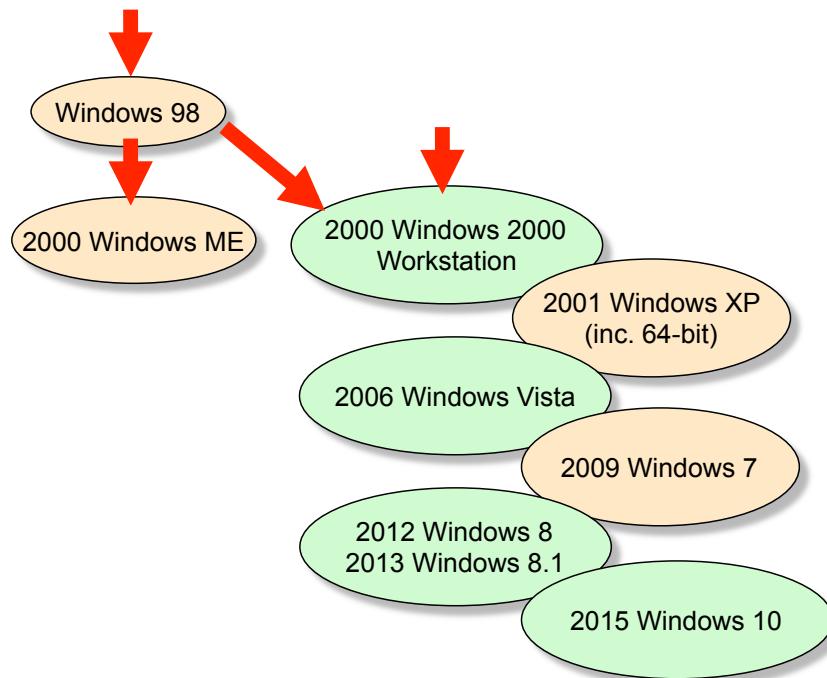
This upgrade to the Microsoft business desktop operating system brought simplified management, higher network throughput, and tools for developing and managing intranets. Windows NT Workstation 4.0 included the popular Windows 95 user interface.

1998: Windows 98

Windows 98 was the upgrade from Windows 95. "Works Better, Plays Better," Windows 98 was the first version of Windows designed specifically for consumers.

With Windows 98, Microsoft added support for reading DVD discs, and support for universal serial bus (USB) devices.

History of 32-bit Windows Desktops



1999: Windows 98 Second Edition

Windows 98 SE, as it was an incremental update to Windows 98. It offered consumers a variety of new and enhanced hardware compatibility.

2000: Windows Millennium Edition

Windows Me was the last Microsoft operating system to be based on the Windows 95 code base.

2000: Windows 2000 Professional

Windows 2000 Professional was designed to replace Windows 95, 98, ME, and Windows NT. Windows 2000 added major improvements in reliability, ease of use, Internet compatibility, and support for mobile computing.

Among other improvements, Windows 2000 Professional simplified hardware installation by adding support for a wide variety of new Plug and Play hardware, including advanced networking and wireless products, USB devices, IEEE 1394 devices, and infrared devices.

2001: Windows XP

With the release of Windows XP in October 2001, Microsoft merged its two Windows operating system lines for consumers and businesses, uniting them around the Windows 2000 code base.

2001: Windows XP 64-bit Edition

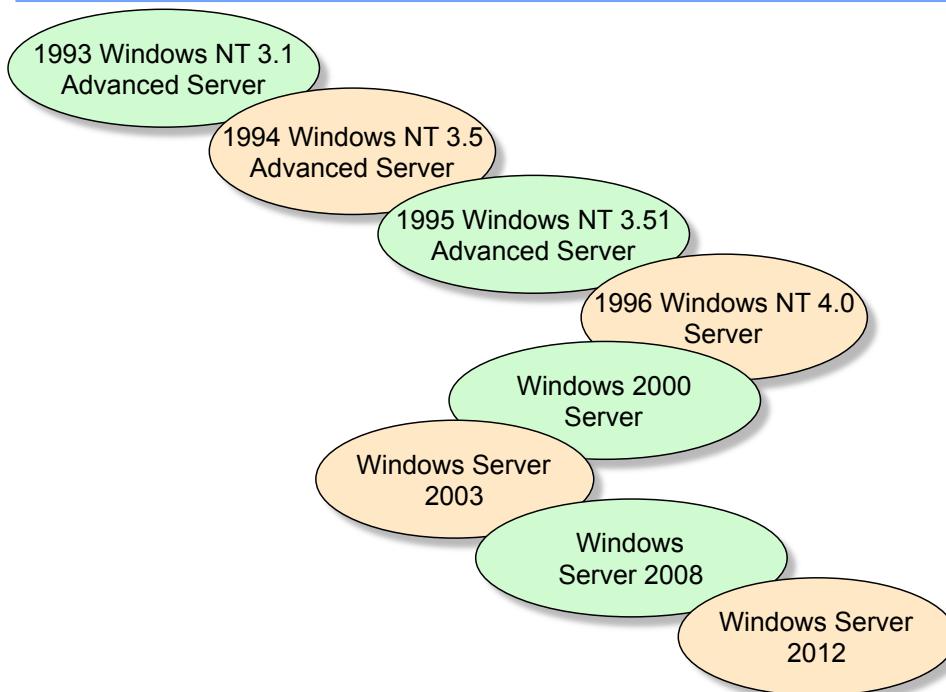
Windows XP 64-Bit Edition satisfies the needs of power users with workstations that use the Intel Itanium 64-bit processor. The first 64-bit client operating system from Microsoft, Windows XP 64-Bit Edition is designed for specialized, technical workstation users who require large amounts of memory and floating point performance in areas such as movie special effects, 3D animation, engineering, and scientific applications.

2006: Windows Vista

Windows Vista introduced significant changes to the Windows security model,

including support for User Account Control (UAC) which reduced the privileges given to user accounts under normal operation, with a means to elevate privileges on demand.²³¹

History of Windows Servers



1993: Windows NT Advanced Server 3.1

Windows NT Advanced Server 3.1 was launched in July 1993 as a dedicated server for a client/server environment.

1994: Windows NT Server 3.5

Enhancements included new administration tools, improved client software configuration, an auto-reboot and dump facility, better tools for NetWare, and better remote access capabilities.

1995: Windows NT Server 3.51

This incremental release of Windows NT Server in June 1995, included a tool to help customers manage Client Access Licenses (CALs).

1996: Windows NT Server 4.0

With this upgrade, Windows NT Server gained the popular look and feel of Windows 95 and added many advanced features for business and technical users.

2000: Windows 2000 Server Family

To support businesses of all sizes, three server versions were offered.

2003: Windows Server 2003

Windows Server 2003 family takes the best of Windows 2000 Server technology and makes it easier and more cost-effective to deploy, manage, and use. The result is a highly productive infrastructure that helps organizations "do more with less."

2003: 64-Bit Operating Systems

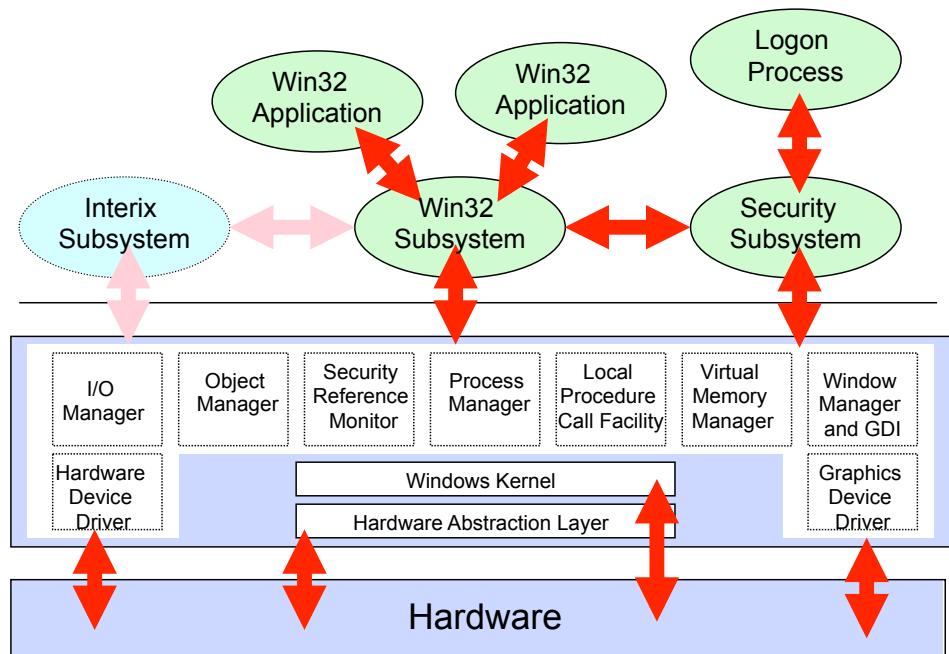
Microsoft features 64-bit versions of all of the Windows Server 2003 family.

2008: Windows Server 2008

Windows Server 2008 shared much of the codebase of Windows Vista. It included .NET Framework 3.0 supporting Windows Communication Foundation (WCF), Windows Workflow Foundation (WF) and Microsoft Message Queuing. It also introduced a new network stack, supporting IPv6 and wireless networking natively.

2012: Windows Server 2012

Windows Architecture



Windows uses two modes, user mode and kernel mode to maintain operating efficiency and integrity.

User mode is where applications and the subsystems that run them are. User mode processes have no direct access to system resources, such as hardware. Resource access requests must be granted by a kernel mode component. User mode components have a lower priority, so they have less access to the CPU cycles than processes that run in kernel mode.

In 64-bit Windows, the Win32 subsystem is replaced with a Win64 subsystem.

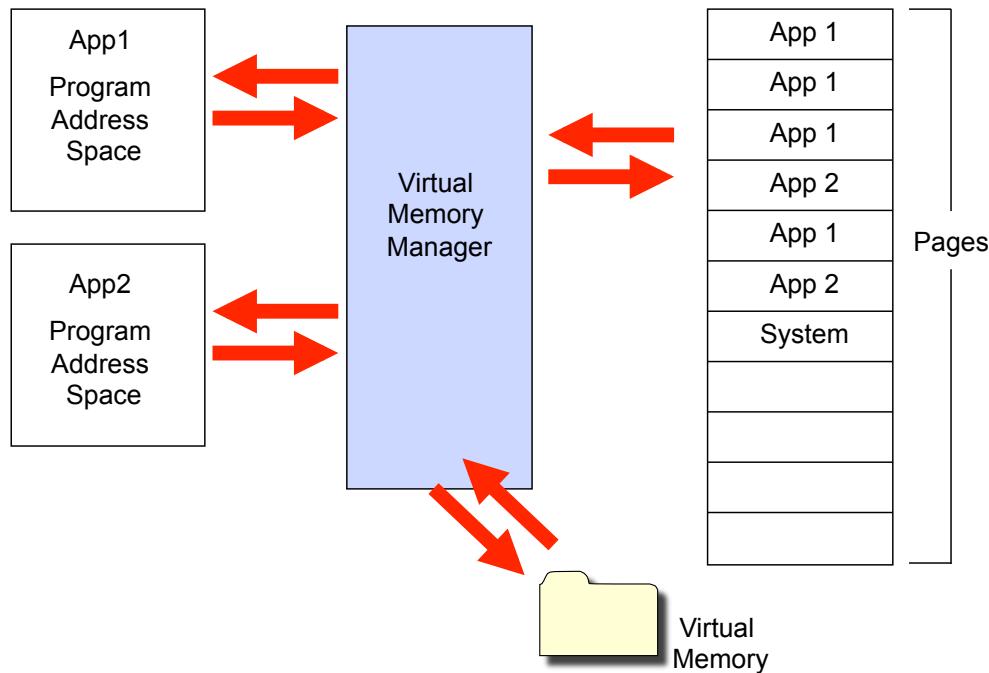
Kernel mode is where the Windows Executive runs. Kernel mode provides access to all of the memory on the computer, and to hardware.

The Executive services consist of managers and device drivers. Managers are the various modules that manage I/O, objects, security, processes, interprocess communications, virtual memory, and graphics management.

The Windows kernel is the service that provides the most basic operating system services such as thread scheduling and interrupt handling.

The Hardware Abstraction Layer contains code that isolates the hardware interface differences from the kernel making Windows portable.

Windows Memory Model



The memory architecture for Windows is demand-paged, virtual memory. It is based on a flat, linear 32-bit or 64-bit address space, which allows each process in Windows to have access to 4GB (or 32GB) of memory.

With virtual memory all applications seem to have access to all memory addresses available. Windows manages this by giving each application a private memory range called virtual memory space and by mapping that virtual memory to physical memory.

Windows maps physical and virtual memory in pages. Although generally the page size is 4KB, this can be set by the CPU. Windows allocates the physical RAM pages between the virtual memory spaces.

The virtual memory process can make use of a paging file on the hard disk. Some of the application code is always kept in RAM whilst other pieces of information can be temporarily paged into virtual memory. When Windows requires the memory that is paged out, Windows reads the memory back into RAM.

Demand paging is the term describing the process of managing which pages are stored in RAM.

1. An application attempts to store data in memory.
2. The Virtual Memory Manager intercepts the request, determines how many pages are needed to fulfil the request, and maps unused physical memory to the application's memory address space.
3. The Virtual Memory Manager may use demand paging to make physical memory available.
4. If the application needs data from the paging file, the Virtual Memory Manager will copy the pages back to RAM.

Devices and Drivers

- Device Driver
 - allows the operating system to talk to a hardware device
- Installing Plug and Play Hardware
 - administrators only install – users can re-attach
 - connect the device
 - detects and configures hardware device
 - installs drivers
 - prompts if input needed
- Installing Non-Plug and Play Hardware
 - connect the device
 - use Add Hardware wizard

For a device to work properly its device driver must be loaded onto the computer. The device driver is software that enables the operating system to communicate with the device. Device drivers are typically supplied by the manufacturer, but there are a number of device drivers included with Windows.

In most cases, it is easy to install hardware on computers running Windows. You simply plug in the new device, which will be automatically detected. The operating system then installs any drivers needed, and updates the system.

To install a Plug and Play device, you must be logged on as Administrator or a member of the local Administrators group.

If a Plug and Play device has been installed and then removed from a computer the device configuration and drivers remain. Reconnecting the same device can be done by a user. This enables users to easily share devices.

For USB, IEEE 1394, SCSI, and other devices that are Plug and Play compliant, just plug in the device. Detection is automatic.

For PCI and ISA Plug and Play cards, unless the hardware manufacturer has PCI Hot-Plug support, turn the computer off, and then install the device. When you restart the computer the device detection and the Plug and Play installation procedures start.

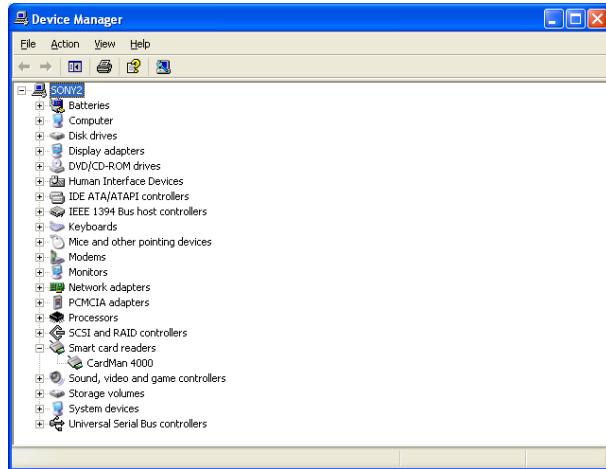
For non-Plug and Play Devices, you connect the device to the appropriate port or slot on your computer. Start the Add Hardware wizard to identify the type of device, then insert the Windows XP Professional CD or the manufacturer's disk so that the appropriate device drivers can be loaded.

After you load the device drivers, Windows configures the properties and settings for the device.

Device Manager

- Device Manager

- displays a “tree” of devices
- displays error icons
- allows device and driver management with right-click



The Device Manager is a utility that allows you to; view a list of installed devices, enable or disable devices, troubleshoot devices, update drivers, and use Driver Rollback.

Device Manager displays a list of the active devices as detected from the registry. This list is recreated each time the computer is started, or whenever a dynamic change occurs. You can expand a node, such as Monitors, and the installed devices under the node will be displayed.

Any bus devices in the tree will have additional device nodes under them. Specific icons indicate the device type and any device conflicts on the computer. If an error state exists, an error icon is displayed.

To update the driver for the device, disable or uninstall the device, scan for hardware changes, or view the device properties, you can right-click the device, and then make your selection on the menu. Double-click the device, and the device's properties sheet is displayed.

Driver Signing

- Signed Drivers
 - verified and tested by signing authority
- “Designed for Windows XX” Logo – Signed by Microsoft
 - passed WHQ lab testing
 - unaltered since the test
- Cryptographic Techniques
 - create .cat file
 - digitally sign .cat file
 - OS then associates .cat file with driver
- Windows XP Driver Signing Options
 - Ignore, Warn or Block

For a device to work properly its device driver must be loaded onto the computer. The device driver is software that enables the operating system to communicate with the device. Device drivers are typically supplied by the manufacturer, but there are a number of device drivers included with Windows.

Windows provides features that make it easy to manage device drivers; driver signing, automatic updates, and driver rollback.

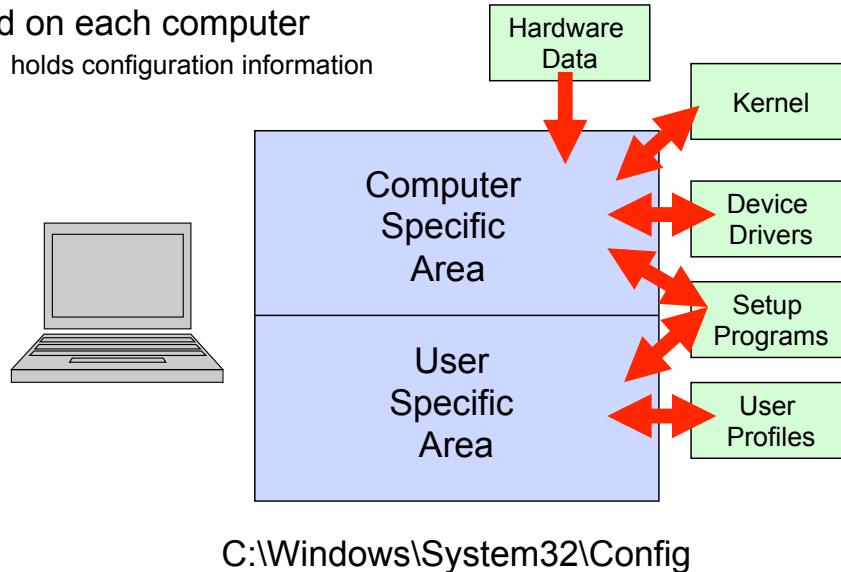
If a driver is "signed" then it has been tested and verified by the signing authority. Microsoft recommends using hardware products that display the "Designed for Windows" logo on the external packaging and on the device itself, as these devices have software that has been digitally signed by Microsoft. This signature ensures that the driver has passed tests administered by the Windows Hardware Quality Lab. And, that it has NOT been altered or overwritten by another program's installation process.

Driver signing uses cryptographic technology to store identifying information in a catalog file. The catalog file is then signed with a digital signature from Microsoft. The relationship between the driver package and its .cat file is maintained by the system after installation.

You can configure driver-signing options to control how Windows will respond if an installation program attempts to add unsigned drivers to the system. The configuration options are; ignore - which installs drivers even if they are unsigned, warn - which will pop a warning message to the installer or, block - which prevents unsigned drivers from being installed.

Introduction to the Registry

- Stored on each computer
 - holds configuration information



The registry is a unified database in which Windows stores all software and hardware configuration information for the local computer. The registry controls the Windows operating system by providing the appropriate initialization information to start applications and load components such as device drivers and network protocols.

The registry helps simplify support by providing a secure structured set of records. It enables administrators to provide local or remote support by using the administrative tools in Windows.

Hardware data is collected during system start up by ntdetect.com and stored in the registry.

The Windows kernel extracts information from the registry, including which device drivers to load and their load order.

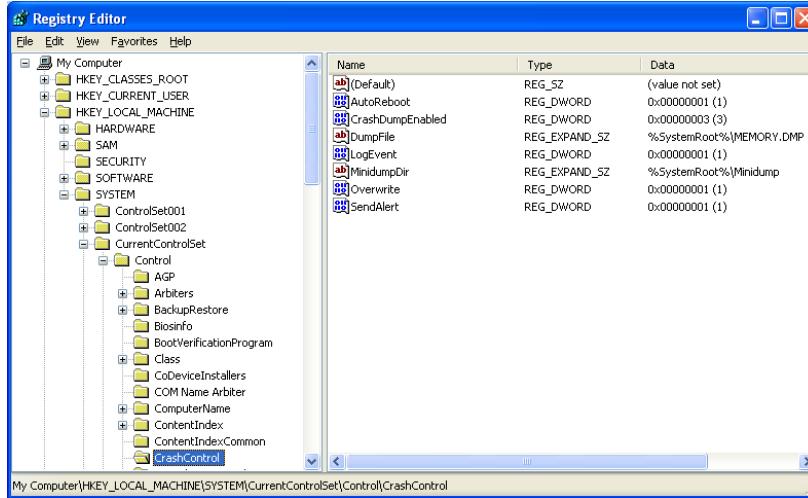
Device drivers pass data to the registry including what system resources it is using and configuration information.

Setup programs add configuration information to the registry. They can also query the registry to see if required components are installed.

User profiles are cached into the registry whenever a user logs on. This ensures that any configuration that a particular user needs is always available for the user.

Registry Structure

- Hierarchical structure made up of “Hives”
 - displayed as trees, subtrees, and keys with values
 - 5 value types



The registry is organized in a hierarchical structure similar to the structure of folders and files on a disk.

The navigation area of Registry Editor displays folders. Each folder represents a predefined key on the local computer. When you access the registry of a remote computer, only two predefined keys appear: HKEY_USERS and HKEY_LOCAL_MACHINE. The following table lists the predefined keys that are used by the system. The maximum size of a key name is 255 characters.

HKEY_CURRENT_USER

Contains the root of the configuration information for the user who is currently logged on. The user's folders, screen colours, and Control Panel settings are stored here. This information is associated with the user's profile. HKEY_CURRENT_USER is a subkey of HKEY_USERS.

HKEY_USERS

Contains all the actively loaded user profiles on the computer.

HKEY_LOCAL_MACHINE

Contains configuration information particular to the computer.

HKEY_CLASSES_ROOT

Is a subkey of HKEY_LOCAL_MACHINE\Software. The information stored here makes sure that the correct program opens when you open a file by using Windows Explorer.

HKEY_CURRENT_CONFIG

Contains information about the hardware profile that is used by the local computer at system startup.

Registry Structure

- “Hives” stored in C:\Windows\System32\Config
 - SAM, SAM.log, and SAM.sav
 - Security, Security.log, Security.sav
 - Software, Software.log, Software.sav
 - System, System.alt, System.log, System.sav
 - Ntuser.dat, Ntuser.log
 - Default, Default.log, Default.sav

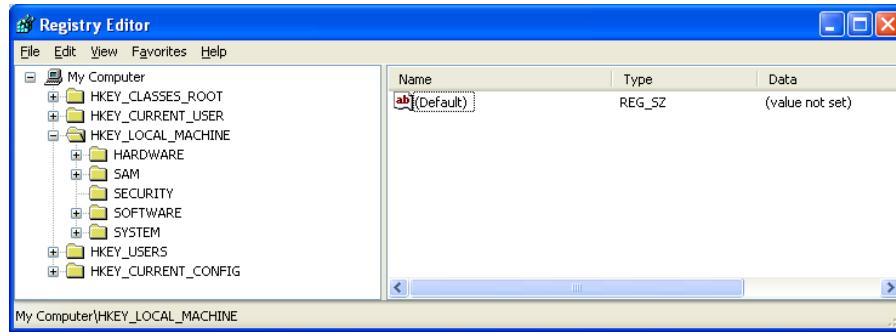


A registry hive is a group of keys, subkeys, and values in the registry that has a set of supporting files containing backups of its data. The supporting files for all hives except HKEY_CURRENT_USER are in the Systemroot\System32\Config folder on Windows NT 4.0, Windows 2000, Windows XP, and Windows Server 2003/2008/2012; the supporting files for HKEY_CURRENT_USER are in the Systemroot\Profiles\Username folder. The file name extensions of the files in these folders, and, sometimes, a lack of an extension, indicate the type of data they contain.

```
HKEY_LOCAL_MACHINE\SAM
Sam, Sam.log, Sam.sav
HKEY_LOCAL_MACHINESecurity
Security, Security.log, Security.sav
HKEY_LOCAL_MACHINESoftware
Software, Software.log, Software.sav
HKEY_LOCAL_MACHINESystem
System, System.alt, System.log, System.sav
HKEY_CURRENT_CONFIG
System, System.alt, System.log, System.sav, Ntuser.dat, Ntuser.dat.log
HKEY_USERSDEFAULT
Default, Default.log, Default.sav
```

Editing the Registry

- Use Regedit.exe
 - direct edits of the registry leave Windows unsupported
 - occasionally the only way to fix a problem



To edit the registry you should follow the steps in the Microsoft documentation only. If you can, use Control Panel instead of directly editing the registry. You can edit the registry by using Registry Editor. If you use Registry Editor incorrectly, you can cause serious problems that may require you to reinstall your operating system. Microsoft does not guarantee that problems that you cause by using Registry Editor incorrectly can be resolved. Use Registry Editor at your own risk.

Before you modify the registry, make sure to back up the registry, and make sure that you understand how to restore the registry if a problem occurs.

Note that the registry in 64-Bit versions of Windows XP and Windows Server 2003/2008/2012 is divided into 32-bit and 64-bit keys. Many of the 32-bit keys have the same names as their 64-bit counterparts, and vice versa.

Some Registry Tips and Tricks

- Add a pre-logon dialog box
 - \SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
 - change the LegalNoticeText value
 - change the LegalNoticeCaption value
- Disable CD-ROM AutoRun
 - \SYSTEM\CurrentControlSet\Services\Cdrom
 - set the AutoRun value to 0
- Don't display the most recent username
 - \SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
 - set the DontDisplayLastUserName value to 1

There are some registry tweaks on the slide to change some standard options in Windows. Use the Internet to find more!

Some Registry Tips and Tricks

- Command prompt from Windows Explorer location
 - HKEY_CLASSES_ROOT \Folder \shell
 - add a new key called CmdPrompt
 - enter the name that you want to display
 - add a key under CmdPrompt named command
 - add c:\winnt\system32\cmd.exe /k cd "%L"
- Change “My Computer”
 - HKEY_CURRENT_USER\CLSID{20D04FE0-3AEA-1069-A2D8-08002B30309D}
 - copy the contents of the key's LocalizedString subkey to Notepad
 - recreate the LocalizedString subkey using REG_EXPAND_SZ
 - copy Notepad's contents into the new subkey
 - change My Computer to %username% on %computername%

There are some registry tweaks on the slide to change some standard options in Windows. Use the Internet to find more!

Default Windows Services

- A Windows service is an application
 - runs in the background
 - managed, started, and stopped by SCM
 - configured with services tool
 - can start automatically, manually, or disabled
 - logs on with Local System, Local Service, or Network Service
- Services provide functionality but beware
 - service may expose functionality to an attacker
 - use principals of least privilege and least service
 - know your system
 - know the default services for your OS

A Service is an application that conforms to the interface rules of the Service Control Manager (SCM). It can be started automatically at system boot, by a user through the Services control panel applet, or by an application that uses the service functions. Services can execute even when no user is logged on to the system.

Because services run unattended at startup, they are well suited to server-type applications such as Web services. However, this characteristic has its drawbacks because a user can be unaware that a service is running. Although some services may open a window or dialog box that is visible to the user, little or no user interaction typically occurs with services, so a user can run a number of default services and never be aware of the potential security risks.

Principle of least privilege. The principle of least privilege states that you give an entity the least amount of access it requires to do its job and nothing more.

Use the Principle of Least Service. The principle of least service states that the OS and any protocols available on any networked device must run only the exact services and protocols required to support the business purpose.

Know your system. To know whether your computers are secure or not, you must know the services that run on your computers and their properties. This information is critical to help keep your servers secure. Consider the value of a table of services and service property settings for services that run on your computers, so that you can immediately evaluate your risks.

Windows at Morgan Stanley

- Standard Desktop Platform

- 6,526 server
 - 63,455 desktop



Whilst Unix has always been the key strategic sever platform at Morgan Stanley, the familiarity and maturity of end-user applications for Windows (particularly Microsoft Office) , has made it the primary desktop.

Windows is also an important server platform, supporting SharePoint, IIS, Microsoft Exchange, windows infrastructure in general, and some business applications.

Currently Deployed Windows Versions

- Windows 7
 - attempts to address many perceived failings in Vista
 - improved UI
 - focus on user productivity
- Key features
 - libraries (logical file grouping)
 - improved taskbar
 - integrated search improvements
 - bundled productivity applications
 - performance and power management improvements
 - touch support
 - integrated virtualisation support (XP mode)
 - new window management capabilities



Windows 7 was released in mid 2009, less than three years after the release of its predecessor, Vista. Windows 7 was intended to be an incremental release and as such contained fewer new features than its predecessor, but focussed on improving the user experience and addressing some of the perceived issues with Windows Vista, particularly in areas relating to user productivity.

Some new facilities were included however, including support for touch devices and integrated virtualisation.

Currently Deployed Windows Versions

- Windows Server 2008
 - web centric
 - virtualization
 - security



Internet Information Services 7.0, which is a modular platform for applications and services, provides a simplified, task-based management interface, greater cross-site control, security enhancements, and integrated health management for Web Services. IIS 7 and .NET Framework 3.0 provide a platform for building applications that connect users to each other and to their data, enabling them to visualize, share, and act on information.

You can virtualize multiple operating systems - Windows, Linux and others - on a single server. Virtualization is built into the operating system with more flexible licensing policies.

Terminal Services Gateway and Terminal Services RemoteApp are designed for easy remote access and application integration with the local desktop.

Network Access Protection enables you to isolate computers that don't comply with your organization's security policies, and provides network restriction, remediation, and ongoing compliance checking.

Federated Rights Management Services provides persistent protection for sensitive data; helps reduce risks and enables compliance; and provides a platform for comprehensive information protection.

Read-Only Domain Controllers allow you to deploy Active Directory Domain Services while restricting replication of the full Active Directory database, to better protect against server theft or compromise.

Windows PowerShell is a new command-line shell with more than 130 tools and an integrated scripting language that enables an administrator to automate routine system administration tasks, especially across multiple servers.

Server Core is a new installation option for selected roles that includes only the necessary components and subsystems without a graphical user interface, to provide a highly available server that requires fewer updates and less servicing.

14 Desktop & Tools

- User Accounts 249
- Logging On 250
- The Desktop 251
- User Profiles 252
- The Control Panel 253
- Windows Command Prompt 254
- File Management Utilities 255
- Process Management Utilities 257
- Miscellaneous Utilities 258
- WinAurora 259
- OpenAFS and Windows 260
- Common Windows Drive Mappings 261
- Local Application Installation 262
- Maintenance and Monitoring 263
- Sysinternals Tools 264
- Process Explorer (procexp.exe) 265
- Process Monitor (procmon.exe) 266
- WinObj (winobj.exe) 267
- Additional Tools 268
- Cygwin 269

User Accounts

- A user account is necessary to logon to Windows
 - name, password, and other configuration options
 - stored in registry or on the network
 - identified by Windows as a globally unique SID
 - permissions and rights associated with SID
 - auditing associated with SID
 - user portion of the registry associated with SID
 - SID's held on Access Token

With Windows, a unique user name and password must be used to log on to a computer. The logon process is mandatory and cannot be disabled.

A user account contains a users unique credentials and enables a user to log on to a computer to gain access to the resources on that computer.

A security id (SID) is a unique numeric string associated with an object, such as a user or a group of users in Windows.

The users SID and any SIDs for groups that the user is a member of, is placed onto an access token by the logon process. Any process that the user launches after logon then contains the access token.

Windows grants or denies access permissions to resources based on access control lists (ACL), which use SIDs to uniquely identify users and their group memberships. When a user requests access to a resource, the user's SID is extracted from the access token and checked against the ACL to determine if that user is allowed to perform the action attempted or if that user is part of a group that is allowed to perform that action.

Logging On

- Local user accounts
 - enable access to local resources
 - reside in SAM
- Domain user accounts
 - enable access to network resources
 - reside in domain
- Built-in user accounts
 - enable administrative tasks
 - provide guest access
 - reside in SAM and domain

A user account contains a user's unique credentials and enables a user to log on to the domain to access network resources or to log on to a specific computer to access resources on that computer.

Local User Accounts

Enable a user to log on to a specific computer to gain access to the resources on that computer.

Domain User Accounts

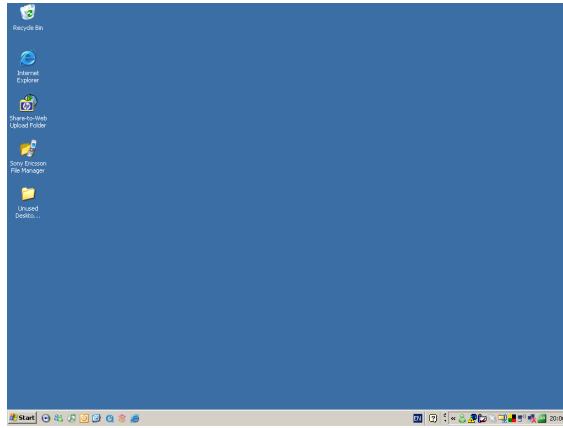
Enable a user to log onto a domain to gain access to network resources.

Built-in User Accounts

Enable a user to perform administrative tasks or to gain temporary access to network resources.

The Desktop

- Loads after logon
 - provides a configurable graphical interface
 - gives access to applications and resources



Users desktops, contained in their profiles, are a configurable and customizable space that can increase user productivity by making frequently used items easily available. You can implement and enforce desktop customization policies by using profiles, which can enable users to gain access to their own desktops from any computer that is on the network.

Some of the advantages of configuring the desktop environment in Windows XP include providing users and organizations that use more than one language the ability to configure desktops for multiple languages and multiple locations. You can also customize the Start menu and taskbar to display the most commonly used programs and network connections.

When you configure user desktop settings, you change the appearance of the work area and the items that it contains.

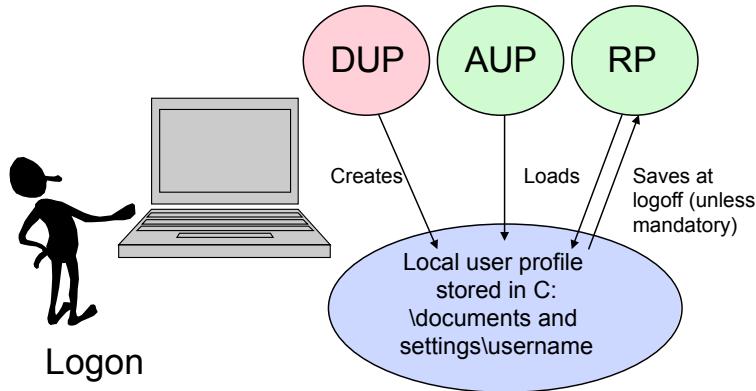
Display properties are used to configure the visual aspects of the desktop, including the background, icons, and fonts.

Themes are a predefined set of icons, fonts, colors, sounds, desktop backgrounds, and other window elements that give your desktop a unified look. You can choose from existing themes, create your own theme by modifying an existing theme, or restore the look used in previous versions of Windows by using the Windows Classic theme.

In addition to enabling you to configure the desktop theme, Display Properties also control the appearance and behavior of windows, buttons, and controls that appear on the desktop.

User Profiles

- User profile types
 - default user profile
 - all user profile
 - local user profile
 - roaming profiles
 - mandatory user profiles



A user profile is created the first time that the user logs on to a specific computer. All user-specific settings are saved in the users profile within the Documents and Settings folder. When the user logs off from the computer, the users profile is updated on that computer. Thus, the user profile maintains the desktop settings for each users work environment on the local computer, unless the profile is mandatory, in which case the user cannot update it.

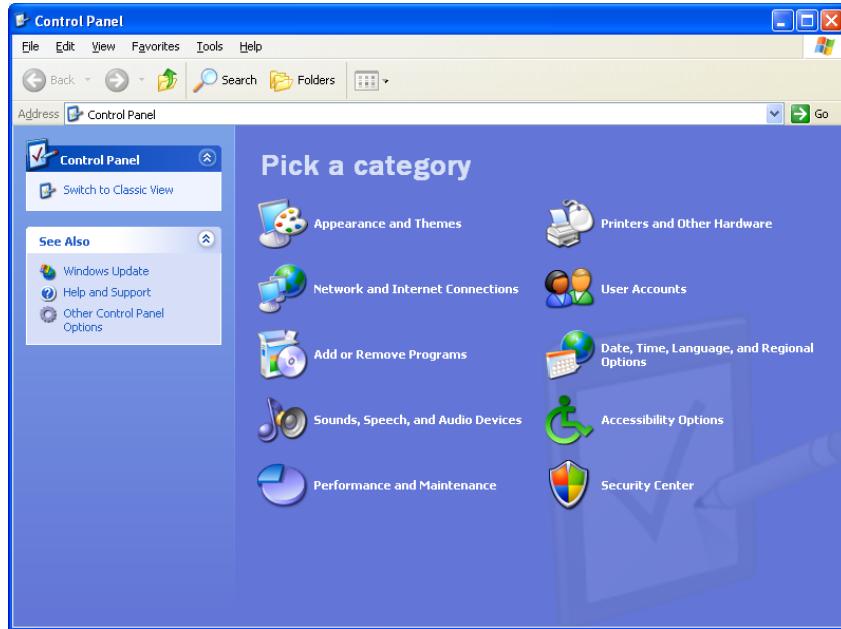
Default user profile. Serves as the basis for all user profiles. Every user profile begins as a copy of the default user profile, which is stored on each computer running Windows.

Local user profile. Created the first time a user logs on to a computer. It is stored on the local computer. Any changes made to the local user profile are specific to the computer on which the changes were made. Multiple local user profiles can exist on one computer.

Roaming user profile. Created by the system administrator and stored on a server. This profile is available every time a user logs on to any computer on the network. If a user makes changes to his or her desktop settings, the user profile is updated on the server when the user logs off.

Mandatory user profile. Created by the system administrator to specify particular settings for a user or users. Roaming profiles can be made mandatory by changing the profile file name from Ntuser.dat to Ntuser.man. A mandatory user profile does not enable users to save any changes to their desktop settings. Users can modify the desktop settings of the computer while they are logged on, but these changes are not saved when they log off.

The Control Panel



System performance can vary over time because of changes in workload and resource usage. Windows contains configuration options that enable you to optimize system performance.

When you configure operating system settings, they apply to all users who log on to the computer; therefore, you do not need to reconfigure the settings for each user. In Control Panel, you can configure the following operating system settings:

Environment Variables. Enables you to alter user and system variables. For example, you can change the location of the system's temporary files to optimize space.

Startup and Recovery. Enables you to configure startup and recovery procedures. For example, you can set the counter to zero to minimize the restart time.

The Control Panel enables you to make many other settings too. For example, you can:

Add or remove programs.

Make date, time, and regional changes.

Configure accessibility options.

Configure security settings.

Windows Command Prompt

- Command line interface for Windows
 - implemented as cmd.exe
 - native windows executable
 - not to be confused with command.com (DOS)
- Important caveat
 - current directory maintained per process, per drive
 - type drive name to change drive or use /d option of CD command

```
C:\temp>cd d:\DOCS  
  
C:\temp>copy d:.*.* c:  
d:WINXP_LOGO_HORIZ_SM.GIF  
    1 file(s) copied.  
C:\temp>dir  
...  
14/04/2008  13:00          6,895 WINXP_LOGO_HORIZ_SM.GIF
```

- in this example, files are copied from d:\DOCS*.* to c:\temp

Many Windows users spend much of their lives without making use of its command line interface. The cmd.exe utility offers an environment, some aspects of which will be familiar to users of Unix-like operating systems. Over the next couple of slides, a brief summary of the available commands (and some of the striking differences when compared to Unix-like environments) are provided.

File Management Utilities

Utility	Description	Comments
chdir (cd)	change directory and drive	can also use <code>pushd</code> and <code>popd</code> to manage stack of directories
dir	directory listing	default output typically more verbose than required
mkdir (md)	create directory	builds required path automatically
rmdir (rd)	remove directory	use <code>/s</code> to remove tree use <code>/q</code> to suppress confirmation prompts
move, copy	move/copy file(s)	similar to their Unix counterparts
xcopy	copy directory trees	maintains relative path between copied files
sort	sorts content	supports alphabetic sorting
rename (ren)	rename file or group of files	often used to change file extension for groups of files

File Management Utilities

Utility	Description	Comments
mountvol	manage volume mount points	can link volumes without using a drive letter
cacls	change discretionary acls	equivalent to Unix chmod and fs setacl commands
attrib	manage file and directory attributes	attributes include hidden, system, archive and read only
type	display contents of text files	similar to Unix cat utility, but adds filename as separator between file contents
del (erase)	remove file(s)	use /s to remove directory tree
comp	compare files	compares two files or sets of files
findstr, find	search files for pattern or expression	equivalent to Unix grep and fgrep utilities respectively
tree	display directory tree	graphically visualises directory tree structure
print, lpr, lpq	manage print queues and print text files	see prnXXX set of utilities to manage printer and print queue configuration

Process Management Utilities

Utility	Description	Comments
taskkill	end one or more processes	can identify processes by name or process id
tasklist	lists active processes	equivalent to Unix <code>ps</code> utility
schtasks	schedule repetitive tasks	equivalent to Unix <code>cron</code> utility
at	schedule one-off tasks	equivalent to Unix <code>at</code> utility
set	change and list environment variables	use <code>setlocal</code> and <code>endlocal</code> to delimit environmental changes to a section of batch file
start	start a new command prompt window	can also be used to launch graphical applications
path	manage path environment variable	alternative to using more general <code>set</code> command
call	call separate batch file	invoke one batch file from within another

Miscellaneous Utilities

Utility	Description	Comments
assoc	manage file associations	relate file types to file extensions and vice versa
bootcfg	manage boot settings	provides abstraction over boot configuration files
cipher	manage file encryption	available on NTFS volumes
compact	manage file compression	available on NTFS volumes
cscript	command line script host	can be used to execute Windows Script Host scripts (typically JavaScript or VBScript)
date	display and set system date	requires administrative access to change system time
help	display command help	with no parameters, displays list of commands

WinAurora

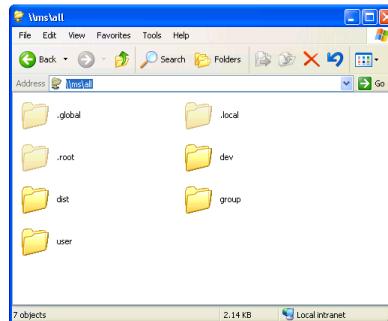
- Project to change Windows platform application model
 - also addresses operational infrastructure
 - in line with concepts of Unix/Linux Aurora platform
- Primary challenges
 1. dependencies on local state
 2. registry settings
 3. hard-coded dependencies on explicit paths
- Solutions
 1. OpenAFS
 2. RegHook
 3. Softricity (vendor product) under evaluation



OpenAFS and Windows

- Provides access to AFS namespace from Windows

- supported on all Windows platforms
 - part of core desktop and IIS Server builds
 - replaces older WinDist (P:\ drive) solution



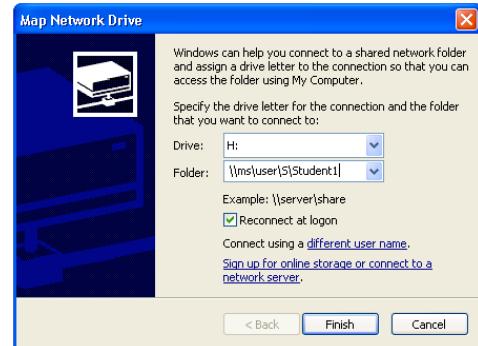
- The module command

- similar to Unix/Linux equivalent
 - useafs and usewindist commands

```
C:\>module load winaurora/compat  
C:\>useafs  
C:\>module load msde/devtools
```

Common Windows Drive Mappings

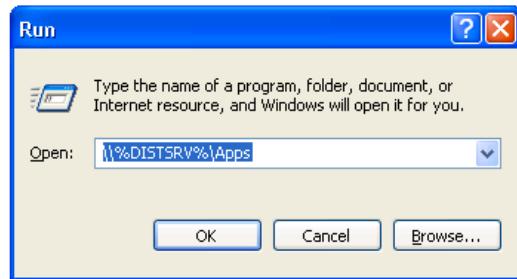
- OpenAFS drive mappings
 - M:\ maps to \\ms\all
 - H:\ maps to \\ms\user\X\<username>
 - use UNC paths unless drive mapping necessary



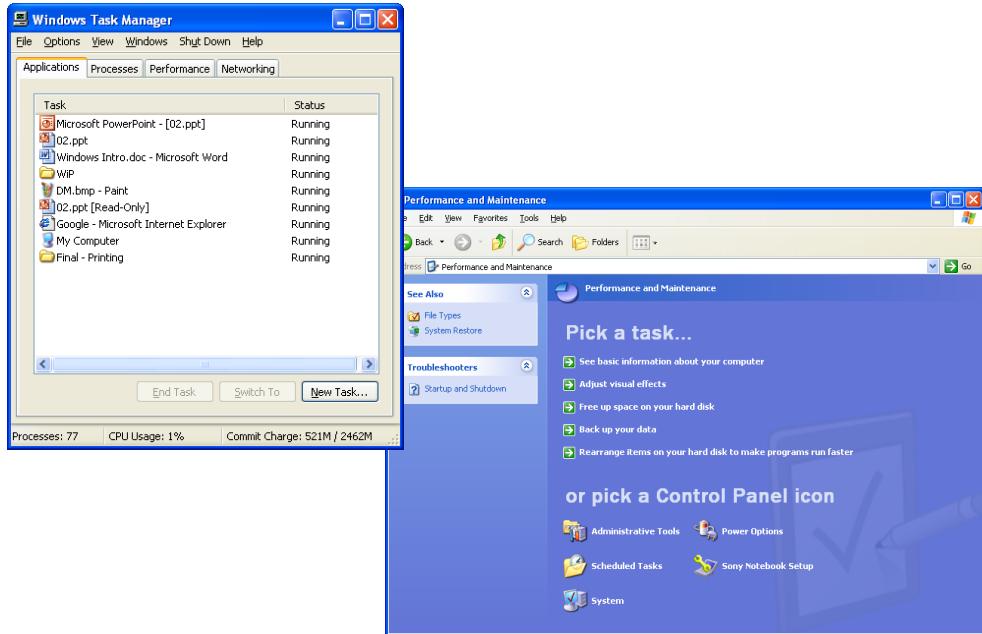
- Additional common Windows mapped drives
 - L:\ maps to standard libraries directory
 - P:\ maps to standard programs directory
 - U:\ may be mapped to Windows user home directory

Local Application Installation

- Distribution servers host install scripts
 - custom go scripts machine installation
 - often dedicated go script to uninstall also
- Finding a distribution server
 - DISTSRV environment variable
 - Apps directory structured as <vendor>/<product>



Maintenance and Monitoring



Task Manager is a tool that provides real-time information about applications currently running on your computer. The available information about applications includes the amount of processor time and memory that the processes associated with the application are using. Task Manager also provides information about the computer's performance and network connectivity.

You can use Task Manager to identify an application or process that is using a disproportionate amount of system resources. In addition, the Task Manager status bar provides you with measurements of system or program activity.

When you choose the Adjust Visual Effects option, the Performance Options property sheet is displayed. The sheet contains two tabs: Visual Effects and Advanced.

The Visual Effects tab enables you to configure visual effects that balance your needs for visual appeal and computer performance. The default settings for the configurable visual effects are based upon your computer's capabilities. You can enable an individual visual effect by selecting its check box, or disable it by clearing the check box. Additionally, you can use three buttons that will automatically configure all of the effects.

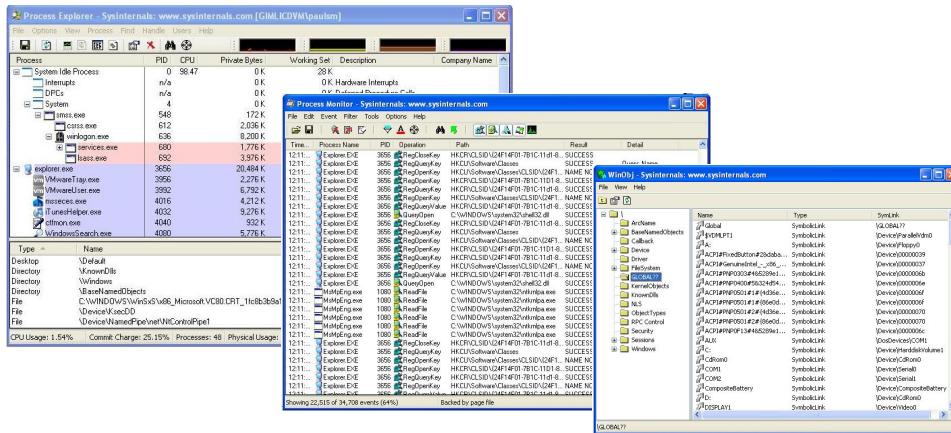
The Advanced tab of the Performance Options property sheet enables you to configure processor scheduling, memory usage, and virtual memory.

You can optimize processor scheduling for either Programs or Background services.

Sysinternals Tools

- Windows troubleshooting and diagnostic toolset
 - originally created by Mark Russinovich and Bryce Cogswell
 - now available through Microsoft TechNet website
 - available in Morgan Stanley through module load

```
C:\>module load msde/devtools
```

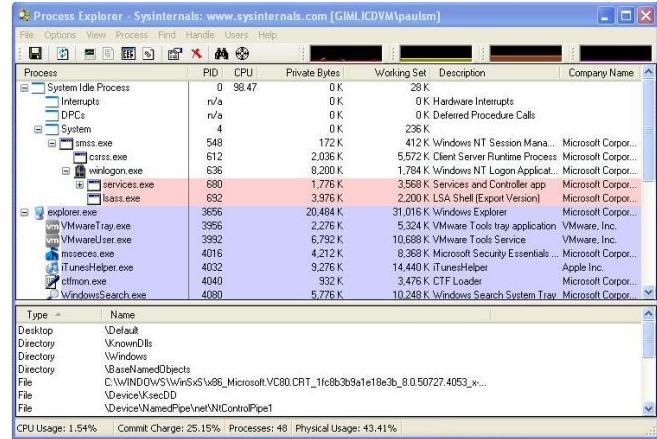


The Sysinternals web site was created in 1996 by Mark Russinovich and Bryce Cogswell to host their Windows system utilities and technical information. The utilities themselves were typically made available for free, some with accompanying source code and articles discussing the APIs and techniques used in the creation of the tools. In July, 2006, Microsoft acquired the Sysinternals site and continued to make the tools available (for free) through their TechNet website (<http://technet.microsoft.com>).

In the Morgan Stanley environment, a subset of the Sysinternals tools is made available through the module load command. You should note that some of the utilities require administrative permissions to execute.

Process Explorer (processexp.exe)

- Graphically presents process tree
 - process resource usage
 - process dependencies
 - debugger and dependency walker tool integration
 - detailed thread information (through process properties)



Process Explorer is a drop-in replacement for task manager which offers a great deal more information on processes, their dependencies and resource usage.

Process Monitor (procmon.exe)

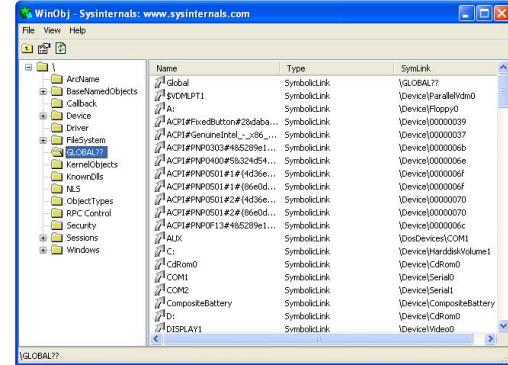
- Monitors most process activities
 - file access
 - registry activity
 - network activity
 - replaces separate tools: regmon, filemon and tcpmon

Time	Process Name	RID	Operation	Path	Result	Detail
12:11....	Explorer.EXE	3656	RegQueryKey	HKEY_CURRENT_USER\Software\Classes\... SUCCESS	SUCCESS	Query: Name
12:11....	Explorer.EXE	3656	RegQueryKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... SUCCESS	SUCCESS	Query: Name
12:11....	Explorer.EXE	3656	RegOpenKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... NAME NOT FOUND Desired Access: Q... Desired Access: Q...	SUCCESS	Desired Access: Q...
12:11....	Explorer.EXE	3656	RegQueryKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... SUCCESS	SUCCESS	Query: Name
12:11....	Explorer.EXE	3656	RegQueryKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... NAME NOT FOUND Desired Access: M... Desired Access: M...	SUCCESS	Desired Access: M...
12:11....	Explorer.EXE	3656	RegQueryValue	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... SUCCESS	SUCCESS	Type: REG_EXPANDABLE
12:11....	Explorer.EXE	3656	QueryOpen	C:\WINDOWS\system32\user32.dll	SUCCESS	Creation time: 14/0...
12:11....	Explorer.EXE	3656	RegCloseKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... SUCCESS	SUCCESS	
12:11....	Explorer.EXE	3656	RegQueryKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... NAME NOT FOUND Desired Access: Q... Desired Access: Q...	SUCCESS	Desired Access: Q...
12:11....	Explorer.EXE	3656	RegOpenKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... SUCCESS	SUCCESS	Query: Name
12:11....	Explorer.EXE	3656	RegQueryValue	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... NAME NOT FOUND Desired Access: M... Desired Access: M...	SUCCESS	Desired Access: M...
12:11....	Explorer.EXE	3656	RegOpenKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... SUCCESS	SUCCESS	Query: Value
12:11....	Explorer.EXE	3656	RegQueryValue	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... SUCCESS	SUCCESS	Type: REG_EXPANDABLE
12:11....	Explor...e	1089	ReadFile	C:\WINDOWS\system32\kernel32.dll	SUCCESS	Creation time: 14/0...
12:11....	MMExec.exe	1089	ReadFile	C:\WINDOWS\system32\kernel32.dll	SUCCESS	Offset: 1,845,456...
12:11....	MMExec.exe	1089	ReadFile	C:\WINDOWS\system32\kernel32.dll	SUCCESS	Offset: 1,828,164...
12:11....	MMExec.exe	1089	ReadFile	C:\WINDOWS\system32\kernel32.dll	SUCCESS	Offset: 1,833,904...
12:11....	MMExec.exe	1089	ReadFile	C:\WINDOWS\system32\kernel32.dll	SUCCESS	Offset: 1,845,044...
12:11....	MMExec.exe	1089	ReadFile	C:\WINDOWS\system32\kernel32.dll	SUCCESS	Offset: 1,845,844...
12:11....	Explorer.EXE	3656	RegCloseKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... SUCCESS	SUCCESS	
12:11....	Explorer.EXE	3656	RegQueryKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... NAME NOT FOUND Desired Access: Q... Desired Access: Q...	SUCCESS	Desired Access: Q...
12:11....	Explorer.EXE	3656	RegOpenKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... SUCCESS	SUCCESS	Query: Name
12:11....	Explorer.EXE	3656	RegQueryKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... NAME NOT FOUND Desired Access: M... Desired Access: M...	SUCCESS	Desired Access: M...
12:11....	EndUserC...	2556	RegOpenKey	HKEY_CURRENT_USER\Software\Classes\CLSID\{24F14F01-7B1C-11d1-8... NAME NOT FOUND Desired Access: M... Desired Access: M...	SUCCESS	Desired Access: M...

Process Monitor provides low level monitoring of all resource usage, including the categories listed above. Given the vast amount of information that can be captured, it is important to become familiar with the filtering facilities offered by the tool.

WinObj (winobj.exe)

- Object Manager namespace viewer
 - displays information on NT Object Manager's name space
 - more accurate than SDK tool of same name
 - supports wider range of object types
 - knows how to open device objects
 - can view and change object security information using native security editors



The Object Manager is a key service of the Windows Executive. WinObj provides a window onto the Object Manager namespace and allows interesting insights into implementation details of the Windows NT-derived operating systems.

For example, as can be seen in the screenshot above, C: is a global object of type SymbolicLink which refers to the object \Device\Harddisk\Volume1.

Additional Tools

- PsTools suite
 - PsExec
 - PsFile
 - PsGetSid
 - PsInfo
 - PsKill
 - PsList
 - PsLoggedOn
 - PsLogList
 - PsPasswd
 - PsService
 - PsShutdown
 - PsSuspend
- DebugView
 - intercepts calls to DbgPrint & OutputDebugString
 - allows viewing/recording of debug session output
 - on local machine or across network
- VMMap
 - process virtual and physical memory analysis utility
- TCPView
 - active socket viewer

There are many other tools made available as part of the msde/devtools module. Some of the more useful ones are listed above.

Cygwin

- Linux-like environment for Windows
 - implements POSIX system call API in terms of Win32 system calls
- Cygwin consists of:
 - DLL (cygwin1.dll) which acts as Linux API emulation layer
 - provides substantial Linux API functionality
 - collection of tools which provide Linux look and feel



There are several ways in which a Unix-like command line environment can be made available on Windows. For example, Microsoft offer a product called Windows Services for Unix, which is the evolution of the original Posix subsystem for Windows NT.

An alternative to this is Cygwin. Cygwin is a Linux-like environment for Windows. It consists of two parts:

A DLL (cygwin1.dll) which acts as a Linux API emulation layer providing substantial Linux API functionality.

A collection of tools which provide Linux look and feel.

15 Application Infrastructure

- Windows Application Types 271
- Dynamic Link Libraries (DLLs) 272
- COM 273
- Interface and Implementation 274
- COM Interfaces 275
- Automation 276
- ActiveX 277
- DCOM 278
- COM+ : COM and MTS 279
- .NET 280
- .NET Application and Component Formats 281

Windows Application Types

- Console applications
 - like Unix applications
 - started from command line
 - standard I/O streams available



- Windows applications
 - based around GUI
 - toolkits available for building GUIs
 - event based programming model



- Services
 - like Unix daemon
 - managed by Service Control Manager
 - may be started/stopped automatically



Although the operating system treats all applications in the same way, it is possible to classify Windows applications into three broad groups.

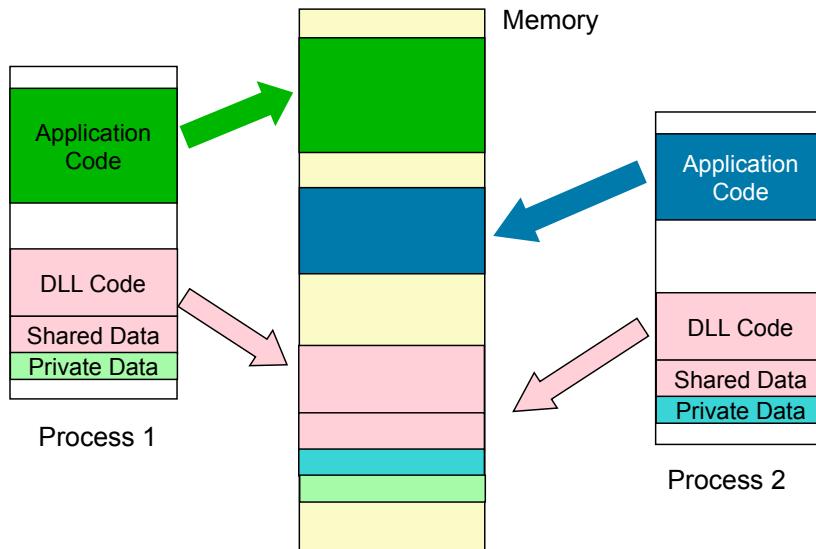
Console applications are the simplest, and equate with the traditional model of application from Unix. The program is normally expected to be run from the command line, and as such has the three standard I/O channels set up for it. A console application may, if desired, start a Graphical User Interface, but normally will not.

Windows applications are the most widespread type of application. They are based around a GUI, which can be built using various programming toolkits and development tools. The most common API toolkits used for building GUIs are MFC and ATL. Windows applications execute using an event based model - once the GUI has been built, further processing is based on "events" such as user input being passed to the application. Events are passed to the application in the form of messages that are enqueued on message queues associated with the window components in the application.

Services are background server programs, similar to Unix daemon processes. They are managed by a separate administrative tool known as the Service Control Manager. Services may be started and stopped manually using the SCM or from the command line using the "NET" command. It is also possible to arrange for services to be started and stopped automatically at system boot/shutdown. Services normally communicate with administrators or users using "events" that are logged and may be viewed using the Event Viewer application.

Dynamic Link Libraries (DLLs)

- Shared application code
 - library functions and data



DLLs provide an efficient implementation of library functionality and other shared application functionality. All shared aspects of the DLL (code and "static" data) are loaded into memory only once, each application that uses the DLL is linked to the in memory copy. Each "client" of the DLL can have its own private instance data.

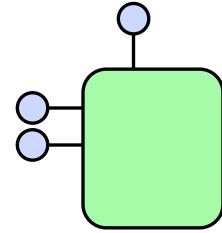
A DLL may be loaded and linked explicitly through API calls or implicitly through loader options set when building the application.

APIs allow for sophisticated initialisation behaviour when a DLL is first loaded into memory or a new process "attaches" to the DLL.

The idea of DLLs was the first step along a road of increasingly sophisticated mechanisms allowing application components to inter-operate with each other.

COM

- Component Object Model
- Binary standard interface specification
 - based on C++ vtable
- Language independent
 - C++
 - Visual Basic
 - C
 - Java
 - etc...



COM is one of the most important Microsoft technologies in recent years.

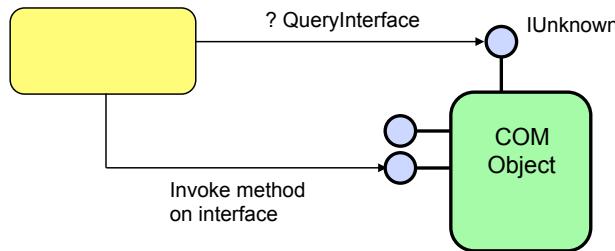
The Component Object Model grew out of work done to overcome difficulties in integrating application components that were written and compiled by different C++ compilers. The name mangling rules for overloaded methods in C++ were not standardised, as a result a reference to a method from one component may not link with the method as defined in a different component.

COM is a binary interface standard that extends the notion of the C++ vtable (used to dispatch virtual method calls). A COM object is the implementation of one or more interfaces.

Because COM is a binary standard, there is no requirement to use a specific programming language. All that is required is the ability to generate code that implements method accessing using the binary standard. As such, COM is considered a language independent component model. The most commonly used languages in a COM environment are C++ and Visual Basic, which presents an easy to use mechanism for accessing and implementing COM objects.

Interface and Implementation

- COM objects may support many interfaces
 - export different services through each interface
- Standard interface *IUnknown* implemented by all COM objects
 - allows discovery of other interfaces



COM is based around the strict separation of interface and implementation. A COM object will provide the implementation of one or more interfaces. Clients of the COM object can only access it through the published interfaces.

One object may provide the implementation of multiple interfaces, each representing a different service that the object provides.

A special interface that should be implemented by all COM objects is called *IUnknown*. The main purpose of this interface is to allow clients of the object to determine which interfaces it supports. A particular method that is available through the *IUnknown* interface is *QueryInterface*, which returns a list of supported interfaces. The client may then use one or more of these interfaces to access the facilities provided by the object.

Reference counting is used to keep track of how many clients are currently accessing a COM object. When the reference count becomes 0, the object may be deleted from the application memory.

COM Interfaces

- Defined using Microsoft IDL
 - variant on IDL as used in DCE, CORBA
- Interfaces identified using UUID
 - stored in registry
- Operations defined as methods
 - return HRESULT

```
[object, uuid(99e34280-4e24-a343-08005ae4381e)]
interface IBank : IUnknown
{
    HRESULT Balance ( [out] double * balance );
    HRESULT Deposit ( [in] double amount );
    HRESULT Withdraw ( [in] double amount );
}
```

Interfaces of COM objects are described in Microsoft's IDL implementation. IDL (Interface Definition Language) was originally used with the DCE (Distributed Computing Environment) and later with CORBA and COM. It is a language that specifies interfaces only - no implementation details are present. The IDL specification is processed ("compiled") into a target language that allows an implementation and client of the interface to access the capabilities.

In COM, every interface is uniquely identifiable using a Universally Unique Identifier (UUID), a 128 bit value that is generated by a supporting tool and is based on location and time data, amongst others...

COM Interfaces and their UUIDs are stored in the system registry.

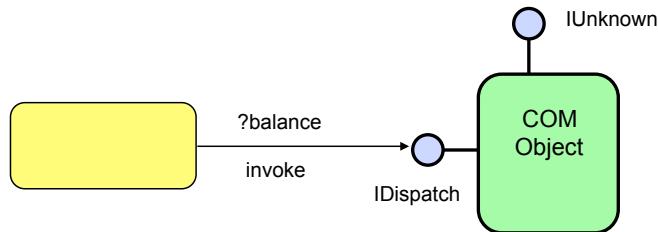
The information contained in the interface definition will mainly be method definitions, although there may be additional type definitions. No data is allowed since this is implementation detail.

All methods in a COM interface return a 32 bit value known as HRESULT - this contains details of the success or failure of the method invocation. Data is passed into and out of the method as parameters - these must be typed and also contain an indication of direction (in, out or inout).

A number of basic types are defined, the definitions are strictly adhered to and contain precise memory sizes - this is essential for cross-language communications.

Automation

- Based around IDispatch interface
 - runtime typing and method invocation
 - designed for Visual Basic
- Supports methods and properties



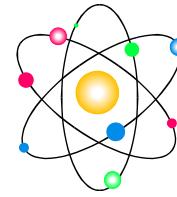
The basic model for using COM relies on compile time checking that the interfaces are being used correctly. Interpreted language environments such as Visual Basic offer the potential of more flexibility if checks can be performed at runtime.

The IDispatch interface forms the basis of a technology known as Automation, where a client can discover details of how to invoke a method, and perform that invocation at runtime.

This model is used extensively with Visual Basic.

ActiveX

- Minimal COM components
 - designed for inclusion in Web pages
 - like Java applets
- IUnknown and IDispatch interfaces
 - components register with OS automatically
- Included in Web page using <object> tag
- Registered on host system after initial download
 - local copy used if subsequent request for the same interface received



ActiveX is a development of COM, whereby stripped down components may be embedded in a Web page. ActiveX was a reaction to the ability to provide "active" content in web pages provided by Java applets.

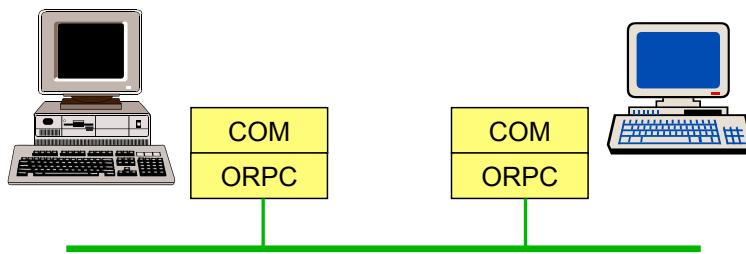
At a minimum, ActiveX components must implement the IUnknown and IDispatch interfaces, these can then be used to access the remaining capabilities.

When an ActiveX component is downloaded (on encountering the <object> tag in an HTML page) it can be authenticated to make sure it is from a trusted source, and then it registers itself on the local system - its UUID being entered into the registry like other COM interfaces. This means that it can be reused from the local system if required again.

Unlike Java applets, ActiveX components are binary objects and therefore are platform specific - they are only available on Windows/Intel systems.

DCOM

- Distributed COM
- Uses Object RPC (ORPC) protocol for transport
 - based on DCE RPC
- Security options available



As described up to now, COM supports communication between components within a single system. As networks and distributed systems became more widespread, DCOM or Distributed COM was developed.

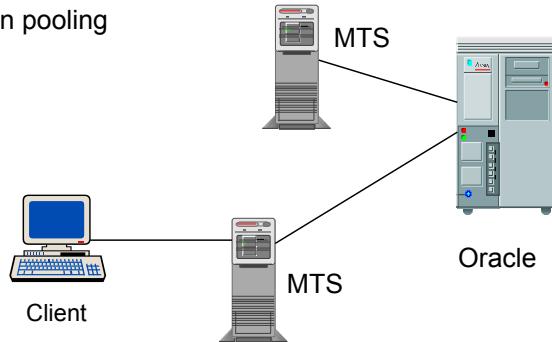
The idea was to layer the method invocations on top of a Remote Procedure Call protocol known as ORPC, derived from the DCE RPC mechanism.

DCOM shares common goals with CORBA - an architecture for distributed component based integration defined by the industry group the OMG.

A number of issues arise when distribution is introduced into an architecture such as COM, one of the most important is security. DCOM provides options for handling security issues, including password protection on method calls.

COM+ : COM and MTS

- Microsoft Transaction Server
 - part of Microsoft DNA (Distributed INternet Architecture)
- Provides middleware facilities
 - transactions
 - thread pooling
 - database connection pooling
 - role based security
 - load balancing
 - queuing

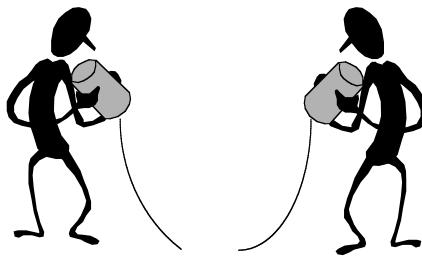


Further support for building large scale distributed applications comes from MTS, which provides middleware capabilities such as resource pooling (thread and database connections) and security through COM objects.

Windows 2000 introduced COM+, a merging of COM and MTS technologies.

.NET

- Combination of technologies to support development of distributed Web based applications
- Core framework
 - Common Language Runtime
 - communications based on SOAP/XML
- Basic Services
 - identity
 - calendar
 - messaging
- Enterprise Services
 - database
 - messaging
 - etc...



The latest approach supporting development of large scale distributed applications is .NET.

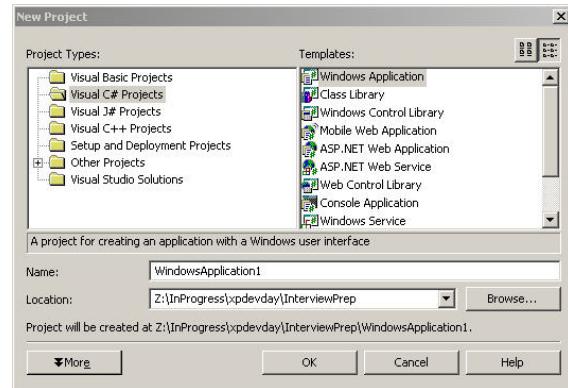
.NET is a collection of technologies built on a common framework, designed around a common execution environment for Windows applications that removes programming language dependence.

On top of this execution framework is a collection of basic services that support usage and application development within the environment. Also available are a number of larger scale "enterprise services" such as database and messaging.

Data is exchanged between the various components and services within .NET using XML, and SOAP is the principal communications mechanism.

.NET Application and Component Formats

- Console Applications
 - standard, text-based applications
- Windows Applications
 - new Windows Forms and GDI+ graphical libraries
- Windows Services
 - standard Win32 Services
- ASP.NET Web Forms
 - document-based web interface
- ASP.NET Web Services
 - XML-based web services
- Various Component types
 - Class libraries
 - Windows Forms controls
 - Web Forms controls



The .NET Framework supports several types of application and component.

Console applications provide support for standard, text-based applications. Full access to the standard input, standard output and standard error streams are available through the `System.Console` class. Windows applications can also be developed using the new Windows Forms and GDI+ libraries. The development of Win32 services is also greatly simplified by the .NET Framework.

Web applications can be developed using the new ASP.NET APIs, which support the development of both document and service-based web interfaces.

Finally, graphical web and windows components, in addition to generic class libraries, can also be developed.

16

End User Computing

- Introducing End User Computing 283
- Migrating to Office 2007 284
- OneNote 2007 285
- Microsoft Office Communicator 286
- Email at Morgan Stanley - Microsoft Outlook 287
- Outlook Productivity and Compliance Tools 288
- Email Safety Catch 289
- Email Signatures and WhereAmI 290
- Microsoft Office LiveMeeting 2007 291
- Windows Desktop Search 292
- Firm Supported Browsers 293
- Remote Access 294
- Introduction to Remote Computing 295
- MyDesk Session Flow for RDP Desktop 296
- Other Products for Remote Computing 297

Introducing End User Computing

- EUC provides the Morgan Stanley infrastructure for:
 - Desktops
 - Laptops
 - Mobile devices
 - Remote computing
- Also provides widely used desktop apps such as:
 - Microsoft Office
 - SharePoint
 - Email
 - Instant messaging
 - Search
 - Business intelligence
- <http://euc>



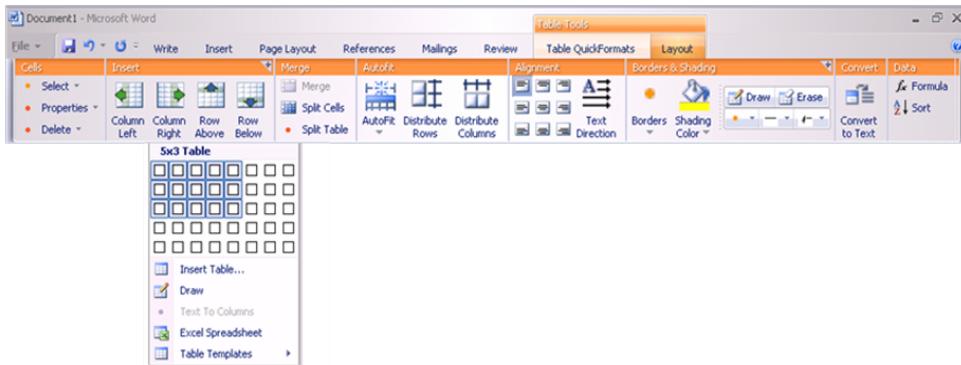
In addition to the desktop infrastructure and user productivity applications, EUC is responsible for externally-facing client technology such as the website platforms for Client Link, MorganStanley.com, and MSToday, as well as report generation and file distribution services.

EUC is functionally organized into the following areas:

Communications, Collaboration, and Archiving
Enterprise Business Intelligence and Distribution Services
Quality Assurance
Windows Team and Remote Computing

Migrating to Office 2007

- Both Office 2003 and 2007 are available
 - 2007 “Ribbon” replaces menus and toolbars
 - Simplifies the choices available
 - Tabs organized around specific scenarios or objects



Office 2007 continues to be an optional offering to business units that believe it will greatly increase their productivity. It is NOT an IT-driven initiative and business units can opt-in or opt-out if they choose. This approach was taken because Office 2003 was fully supported by Microsoft (until April 2009) and the costs of upgrading to 2007 (during a difficult business climate) were significant (e.g. desktop hardware upgrades, etc).

To date, there has been broad interest across the Firm, but only Investment Banking has committed to a global deployment. We also have many other business units (FID, Equities, Research, Operations) piloting/testing Office 2007 as a published application on Citrix.

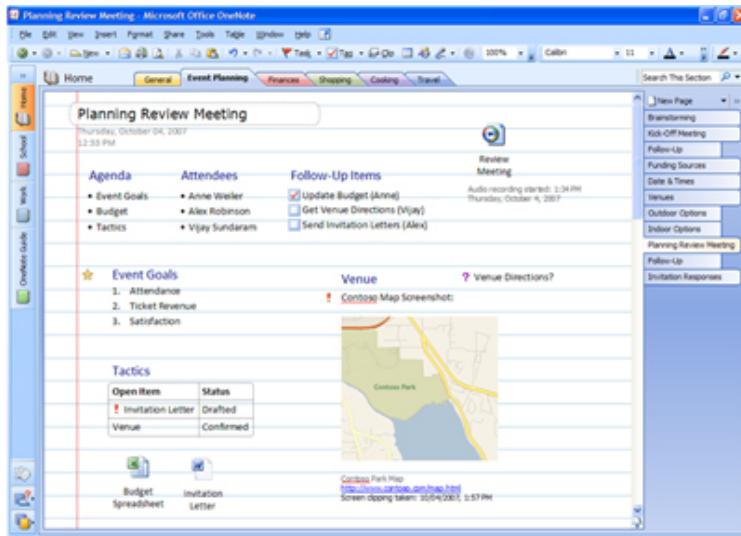
The core Office 2007 build is fully supported, but users should consult their local BU support team to confirm compatibility with line of business products and services.

Microsoft has just entered Office 2010 beta and it is expected to be released in Q1/2010 as part of the wave of v14 products. We have already begun conversations with key business decision makers and teams in the IDEAS organization to discuss the benefits of this release and to formulate a joint strategy. Additionally, we are reviewing other planned deployments for 2010/2011 (Virtualized Desktops, Windows 7, etc) to see where it makes sense to bundle.

Office 2010 is currently in Beta, with an RTM date of March 2010 by Microsoft. The EUC Office Engineering team is targeting July 2010 to have a test production build to allow BUs and IDEAS teams to test and remediate Office dependant applications, with Firmwide Office 2010 deployment expected to start at the end of 2010 and the beginning of 2011.

OneNote 2007

- A personal project management and productivity tool
 - For note-taking and group collaboration



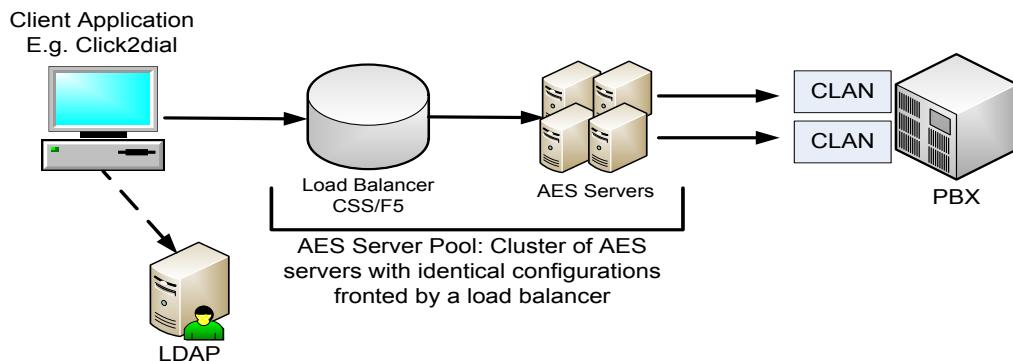
Microsoft Office OneNote is a personal project management and productivity tool for note-taking and group collaboration. As an integrated part of the Microsoft Office system, OneNote makes it easy to gather, organize, find, and share your notes and information more efficiently and effectively. Powerful search capabilities can help you locate information from text within pictures, or from spoken words in audio and video recordings. Easy-to-use collaborative tools help teams work together with all of this information in shared notebooks, whether online or offline. Plus, the familiar look and feel of the Microsoft Office system makes it easy to start using the program right away, minimizing wasted time and training costs.

The Microsoft Office OneNote application is optionally available to members of the Institutional Securities Group (ISG) with a separate license purchase

Please Note: installation of this requires the Office Core Package.

Microsoft Office Communicator

- A unified communications application
 - Enables users to communicate and collaborate easily with others
 - In different locations or time zones
- A range of different communication options, including:
 - Instant messaging (IM), Voice / Video, desktop sharing



Office Communicator is a unified communications application that helps end users be more productive by enabling them to communicate and collaborate easily with others in different locations or time zones using a range of different communication options, including instant messaging (IM), voice, desktop sharing and video. Integration with programs across the Microsoft Office system.

Office Communicator 2007 is the latest messaging and collaboration product from Microsoft, replacing Office Communicator 2005.

Features of Office Communicator include:

Presence indicator : Provides detailed information about a contact's availability and status based on the contact's Outlook 2007 Calendar, login status, and other information sources.

Instant Messaging: Start and receive instant messages from users within the Morgan Stanley network and invite more users to join the conversation. Group Chat has also been incorporated into Communicator.

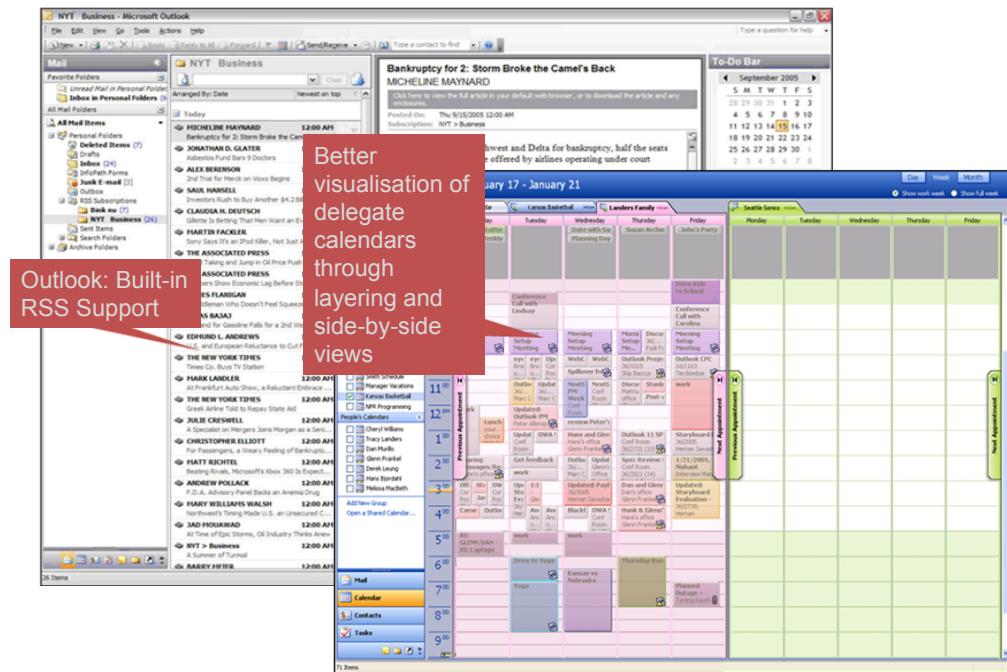
Video Conferencing: Make point-to-point video calls directly from Microsoft Office Communicator without opening another application. Refer to the Video Conferencing Section for more information. A QRC is available with information regarding setup.

Click-to-Dial: Make calls directly from Microsoft Office Communicator without dialling manually.

Sharing with Live Meeting: While instant messaging, making an audio call, or video conferencing, have the option of sharing your desktop with Live Meeting directly from the MOC IM client.

For more information on Microsoft Office Communicator, refer to the QRCs.

Email at Morgan Stanley - Microsoft Outlook



Email in Morgan Stanley is treated as a business critical application. The operations team provides a global 24x7 (follow the sun) support model. The email infrastructure includes the perimeter plant (aka MSA/MTA plant) which consists of Postfix servers / Symantec Brightmail appliances and the Exchange 2007 plant which hosts mailboxes for Outlook users. The infrastructure and operational support model is designed to provide a highly available, fast and resilient service to the business.

Microsoft Outlook is a comprehensive time and information manager that integrates email and calendar functions that include contact and task management. When used in conjunction with Microsoft Exchange Server, Outlook provides enhanced functions for multiple users in an organization, such as shared mailboxes and calendars, public folders and meeting time allocation.

Outlook Productivity and Compliance Tools

- AttachLink
 - A plug-in which allows users to send large attachments
 - Up to 75MB via email or meeting request to internal recipients
 - Uploads files to a central repository
 - Attachments replaced with smaller files - links to the actual files
 - Attachments will expire in 31 days
- Business Information Barrier
 - Restricts flow of e-mail between IBD and ER
- A given user in a restricted business unit will either be:
 - Completely block of messages to users in opposing business unit
 - Will be free of any restrictions

AttachLink is a Microsoft Outlook plug-in which allows users to send large attachments (up to 75MB) via email or meeting request to internal recipients.

AttachLink allows the files to be uploaded to a central repository. The message's attachments are replaced with much smaller files which contain links to the actual files. These can be opened by the recipient simply by double-clicking on them. The result is much smaller messages which will help significantly reduce the size of your mailbox and recipients' mailboxes. AttachLink also allows the sending of much larger files via email than can be sent as regular attachments. The attachments will expire in 31 days, which means the attached file will only be accessible from the archive after 31 days.

For more info refer to <http://attachlink>

In order to meet requirements set by the SEC Morgan Stanley has decided to implement a messaging based Business Information Barrier. The primary purpose of this barrier is to restrict the flow of e-mail messages between two of Morgan Stanley's business units: IBD (Investment Banking Division) and ER (Equity Research). Due to the implementation complexities inherent in deploying rules based control system of this nature the targeted production solution will not permit granular levels of messaging restriction. A given user in a restricted business unit will either be completely blocked from sending messages to users in the opposing business unit or will be free of any restrictions.

Email Safety Catch

- ESC for the Desktop is a Microsoft Outlook add-on
- Looks for exceptions
 - Prompts when recipient is external
 - Helps to prevent accidentally sending emails to the wrong recipient
 - Also sending sensitive data or “Internal Use Only”
 - Leaving Subject line empty
 - Ultimately BU specific list of exceptions
- When ESC detects a potential misdirect condition
 - It warns and prompts you for corrective action
 - Based upon the ESC rules enabled for your business unit
 - ESC uses predefined rules to detect possible misdirect conditions

Email Safety Catch (ESC) for the Desktop is a Microsoft Outlook add-on that helps prevent internal users from accidentally sending emails to the wrong recipient. When ESC detects a potential misdirect condition, it warns and prompts you for corrective action. The conditions detected and the corrective actions available to you are based upon the ESC rules enabled for your business unit. ESC uses predefined rules to detect possible misdirect conditions.

When you click Send, ESC runs the rules activated for your profile, looking for exceptions. For example:

ESC can help prevent unintended external misdirects by prompting you to confirm external email addresses found in the To, CC, and BCC fields.

ESC can help prevent internal misdirects by prompting you to confirm an internal recipient if it finds other employees with similar names.

ESC also can also prevent you from accidentally sending sensitive materials to external recipients. It does so by checking if the email you are forwarding, or an attached document, has been tagged for internal use only.

ESC blocks external email distribution of emails and documents tagged "For Internal Use Only." If you are authoring the email and the information contained in the email is sensitive, the ESC iTag feature enables you to tag the document as "Internal Use Only." This prevents internal recipients from forwarding your email to external addresses.

ESC can also check for common authoring errors such as leaving the email Subject line blank or forgetting to attach a file referenced in the body of the email.

For a complete list of the Email Safety Catch rules. For a complete list of ESC rules refer to <http://escrules>

Email Signatures and WhereAml

- Standardization of Morgan Stanley e-mail signatures
 - Professional, branded appearance for all e-mail communications
 - Keeping your contact information up-to-date

Mary Sample, Vice President
Morgan Stanley | Technology
1 New York Plaza, 9th Floor | New York, NY 10004
Phone: +1 212 276-8950
Mary.Sample@morganstanley.com

- WhereAml facility helps you notify your whereabouts
 - To both internal and external users
 - Gives selective control over various channels of communication
 - For while you are away from your normal working location

In order to protect and reinforce Morgan Stanley's unified brand and to support regulatory and compliance requests, the Firm has launched an initiative to standardize all Morgan Stanley e-mail signatures. The goal is to provide a professional, branded appearance for all firm e-mail communications while automatically keeping your contact information up-to-date. For more information, refer to <http://emailsig>

The WhereAml facility <http://whereami> helps you to inform both users internal to Morgan Stanley as well as external users of your whereabouts. It does so by allowing you to selective control the following channels of communication while you are away from your normal working location.

Have Firmwide Directory display a personalized message next to your name when your colleagues look you up in the Firmwide Directory

Have your Email system generate automatic replies to your colleagues who sends you emails internally

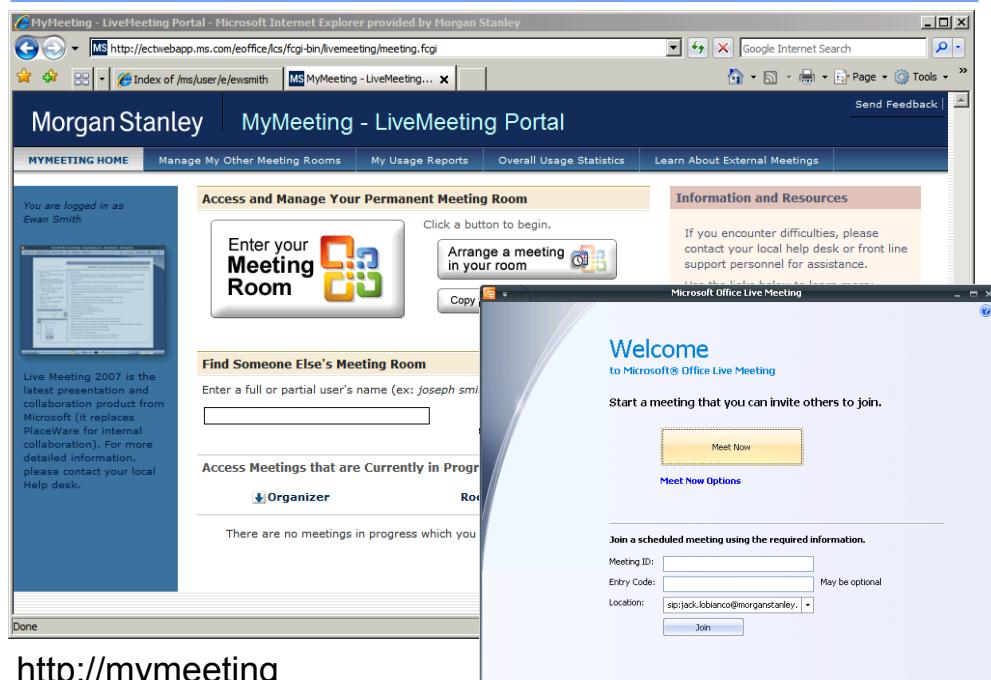
Have your Email system automatically reply to user external to Morgan Stanley who sends you emails

Have the Voice Mail system's Virtual Operator (internal callers pressing '0' on their phones, or dial 3-677-11 11 or 3-762-0 00) announce that you are on vacation and ask if the caller still wants to leave you a voice mail.

The WhereAml tool is automatically launched when you click on Tools > Out of Office Assistant in Outlook.

For more information on other Outlook applications and plug-ins, refer to <http://outlook>. Refer to the Search Archive Tool in the 'Find' section for more information on searching archived mail.

Microsoft Office LiveMeeting 2007



<http://mymeeting>

Microsoft Office Live Meeting 2007 is a hosted conferencing application/service that connects and engages audiences in online meetings, training, and events through Office Communications Server 2007 service. With meeting attendees participating from their PCs, you can deliver a presentation, kick off a project, brainstorm ideas, edit files, and collaborate on whiteboards.

User Logon and SIP Server Authentication is handled by Office Communicator User and Power User Policy Logon Script. This will allow the user to launch the application and authenticate to the OCS 2007 Infrastructure based on membership to these Group Policies.

On a user's first launch of this application, the Live Meeting Console will Launch with the "Meet Now" Button available for scheduling meetings as a presenter:

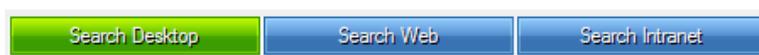
The Location field listed in the screenshot keeps a listing of all attended meetings and can be used to join specifically scheduled meetings.

The following are the components in place for launching, scheduling and hosting Live Meeting 2007 meetings: Live Meeting 2007 is only supported for internal hosting and connectivity to the Office Communications Server 2007 Infrastructure. This requires membership to the Office Communicator User/Power User Group Policy, and an OCS Account registered with the OCS2007 Server Infrastructure.

Points of Failure: a user is not receiving the Office Communicator User/Power User Logon Script and Group Policy; a user does not have an OCS Account; a subnet within the Morgan Stanley infrastructure is down.

Windows Desktop Search

- Windows Desktop Search (WDS)
- Searches for:
 - Outlook messages
 - Documents in your My Documents folder from the user's desktop
- Can also use the accompanying Outlook live toolbar
 - To perform searches from Intranet or Web



For more information, see the QRC, FAQ and DAO Support sites from the course wiki page.

Firm Supported Browsers

- IE11

- tightly coupled with Windows
- local files typed in IE11 are opened using the Windows Explorer
- native Tabbed Browsing
- customizable Drop-down Search



- Google Chrome

- broad standards support
- rich development toolset



- Firefox

- improved standards support
- tabbed browsing and spell checker
- integrated download manager and search system
- testing toolset (Selenium, FireBug)



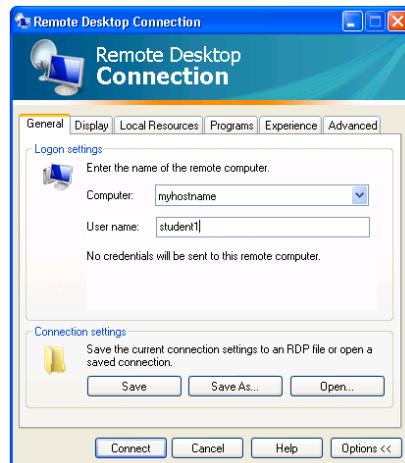
Microsoft Internet Explorer (abbreviated MSIE or IE), is a graphical web browser developed by Microsoft Corporation and is currently one of the most popular web browsers in use today. Internet Explorer 11 is the browser standard within Morgan Stanley on Windows 7.

Google Chrome has also been deployed as an alternative choice to Internet Explorer. It offers significantly better standards support for technologies including CSS3 and HTML5. It also provides a rich set of development tools, including a powerful debugger.

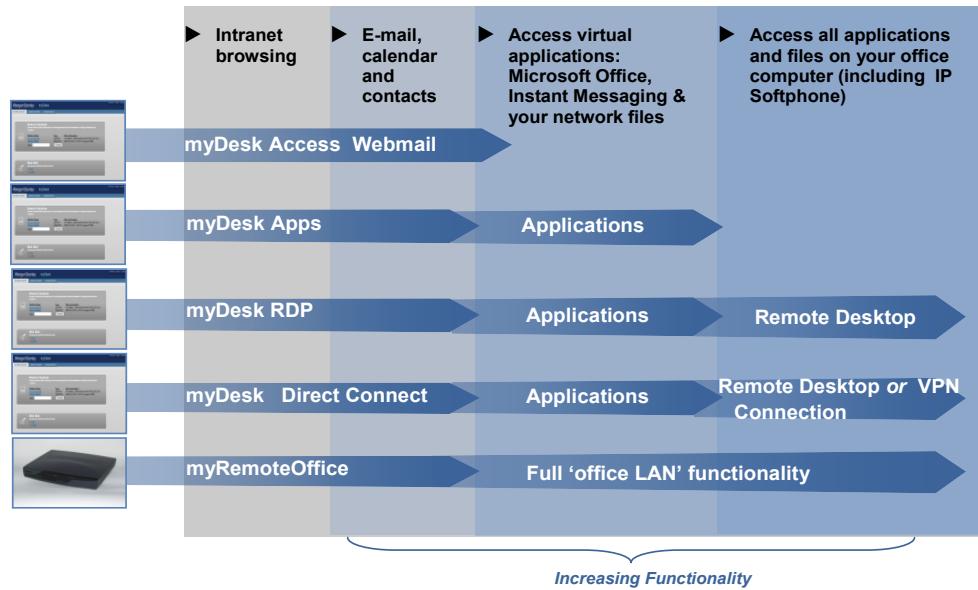
Historically, FireFox has also been available as a BCP browser option. It has a rich set of plugins available within the firm and is still used by many developers due to prevalence of key tools such as Selenium and FireBug.

Remote Access

- Remote Desktop Connection
 - tool to manage remote sessions with Windows machine
- Windows XP supports only one session at a time
 - connection not possible if machine in use
 - administrator may kill existing sessions
- VM slices
 - increasing efforts to virtualize desktops
 - many virtual machines may run on a single server



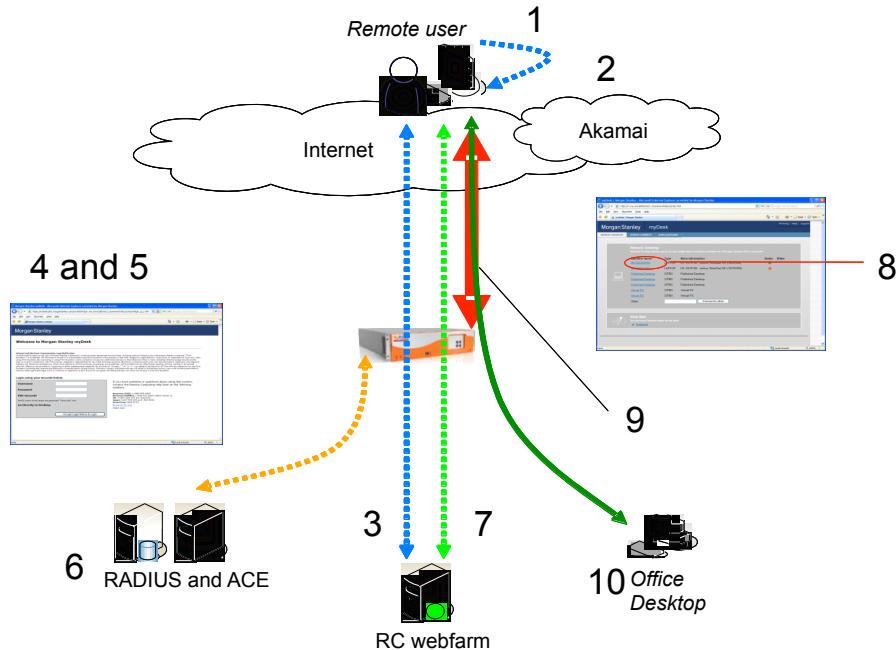
Introduction to Remote Computing



Our Remote Connectivity products are designed to allow you to work normally when out of the office, and depending on the product you are using, provide compatibility with nearly all types of remote machine (Home, Morgan Stanley supplied and Public computers). They also allow you to connect to the Firm via most Internet based access mediums, including standard Wired Broadband (DSL and Cable), Wireless Broadband (Wi-Fi access points) and Mobile Wireless Networks (3G and EVDO).

Our most versatile connectivity product is myDesk and will generally be your first choice for working remotely. It is compatible with Home and Morgan Stanley supplied machines and allows you to remotely access your normal office (or virtual) PC with all its applications and network folders, connect to the Morgan Stanley network (Morgan Stanley machines only), or use Citrix versions of the most popular applications you would generally use when in the office.

MyDesk Session Flow for RDP Desktop



1. User requests <https://mydesk.morganstanley.com>
2. Akamai service response with IP of a regional SPX based on geographic mapping and liveness. Redirect to site specific e.g. <https://mydesk-pi01.morganstanley.com>
3. Retrieve login page from webfarm
4. Check browser user agent string to differentiate managed laptop from unmanaged machine. Run host check on managed laptop to check for FVE, bypass host check on unmanaged machines
5. Redirection to appropriate myDesk site depending on managed or unmanaged access level
6. Authenticate user w/ SecureID token and record session accounting information
7. Present myDesk landing page with list of user's machines as defined by Activeview for ISG, Asset Center or GWM, and Citrix.
8. User selects RDP target machine
9. Array ActiveX transport control starts, creating an SSL connection to the Array SPX.
10. SPX establishes an RDP connection to the target desktop. Microsoft control on client machine provides a display for the RDP session and connects through the Array transport control

Other Products for Remote Computing

- SecurID
 - If you wish to login to the Firm remotely
 - You will first need to request a SecurID
 - A Key Fob or SoftID if you are a BlackBerry user
- MyRemoteOffice
 - A premium "home office" solution
 - Resembles normal Morgan Stanley desk
- A Firm supplied Laptop and Avaya IP Hardphone
 - Via a standard broadband connection at home
- IP Hardphone
 - Your Avaya desk phone at home, most of the standard features

SecurID If you wish to login to the Firm remotely, using one or more of our Remote Connectivity products.

MyRemoteOffice - is a premium "home office" solution , providing a configuration that closely resembles your normal Morgan Stanley desk in terms of hardware, functionality, usability and productivity.

By leveraging the latest Cisco VPN Hardware and an "always on" connection to the Firm's network you will be able to use a Morgan Stanley supplied Laptop and Avaya IP Hardphone via your standard broadband connection at home (Note: An option for a home network printer will also be available to some users).

IP Hardphone - IP Hardphone effectively gives you your Avaya desk phone at home with most of the standard features, including multiple call appearances, speakerphone, conference calling and voice mail notification and retrieval.

QRCs & Quick Facts (the URLs for these are on the course wiki page).

17

Introduction to Mainframes

- Mainframe Operating Systems 299
- Direct Access Storage Device 300
- Data Sets 301
- Mainframe Job 302
- Morgan Stanley Mainframe Environment 307
- MSSB Mainframe Environment 308
- ISG Mainframe Environment 309
- ISG Mainframe Storage 310
- ISG Mainframe Tape 311
- Mainframe Environments 312
- Z/OS 313
- What is ADABAS? 314
- Natural Language 315

Mainframe Operating Systems

- Five characteristics of mainframe operating systems:
 - virtual storage (VMS)
 - multiprogramming
 - spooling
 - batch processing
 - time sharing



Some argue that virtual machines as we know them today in Vmware and other hypervisors existed in the 1970s with mainframes. Many of the same concepts apply such as virtual storage which is the presentation of physical storage in a logical manner abstracted from the hardware.

Mainframes also employ the concept of time sharing, which shares the underlying CPU cycles to various programs in the environment, providing computing power based on scheduling algorithms.

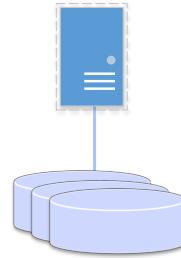
Mainframes also allow multiple types of programs to operate on top of the OS allowing businesses to write in various languages suitable to their applications

Mainframes employ batch processing for large compute tasks (i.e. processing end of day files for investments) and these series of tasks are put into a batch job which can take a few minutes or several hours.

Spooling is used to stream I/O into various devices and programs within the mainframe environment.

Direct Access Storage Device

- IBM term for a disk drive
- Called DASD for short
- Operating system handles read and write on disk packs
- Storage capacity varies based on:
 - tracks
 - cylinders



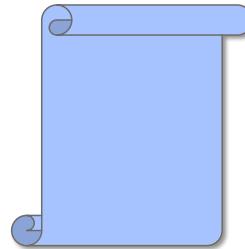
DASD, pronounced DAZ-dee (Direct access storage device), is a general term for magnetic disk storage devices. The term has historically been used in the mainframe and minicomputer (mid-range computer) environments and is sometimes used to refer to hard disk drives for personal computers. A redundant array of independent disks (RAID) is also a type of DASD.

The "direct access" means that all data can be accessed directly in about the same amount of time rather than having to progress sequentially through the data.

IBM mainframes access I/O devices through 'channels', a type of subordinate mini-processor. Channel programs write to, read from, and control the given device.

Data Sets

- Mainframe data stored in form of Data Sets
- VSAM and non-VSAM data set organizations
- VSAM
 - key sequenced
 - entry sequenced
 - relative record
 - linear record
- Non VSAM
 - physical sequential
 - index sequential
 - direct organization
 - partitioned organization



Virtual storage access method (VSAM) an IBM disk file storage access method. Originally a record-oriented filesystem, VSAM comprises four data set organizations: Key Sequenced Data Set (KSDS), Relative Record Data Set (RRDS), Entry Sequenced Data Set (ESDS) and Linear Data Set (LDS). The KSDS, RRDS and ESDS organizations contain records, while the LDS organization (added later to VSAM) simply contains a sequence of pages with no intrinsic record structure, for use as a memory-mapped file.

VSAM records can be of fixed or variable length. They are organised in fixed-size blocks called Control Intervals (CIs), and then into larger divisions called Control Areas (CAs). Control Interval sizes are measured in bytes - for example 4 kilobytes - while Control Area sizes are measured in disk tracks or cylinders. Control Intervals are the units of transfer between disk and computer so a read request will read one complete Control Interval. Control Areas are the units of allocation so, when a VSAM data set is defined, an integral number of Control Areas will be allocated.

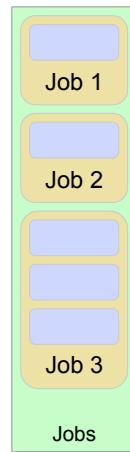
The Access Method Services utility program IDCAMS is commonly used to manipulate ("delete and define") VSAM data sets.

Custom programs can access VSAM datasets through Data Definition (DD) statements in Job Control Language (JCL), via dynamic allocation or in online regions such as in Customer Information Control System (CICS).

Both IMS/DB and DB2 are implemented on top of VSAM and use its underlying data structures.

Mainframe Job

- Execution of one or more related programs in sequence
- A collection of related job steps
- A job is identified by a job statement



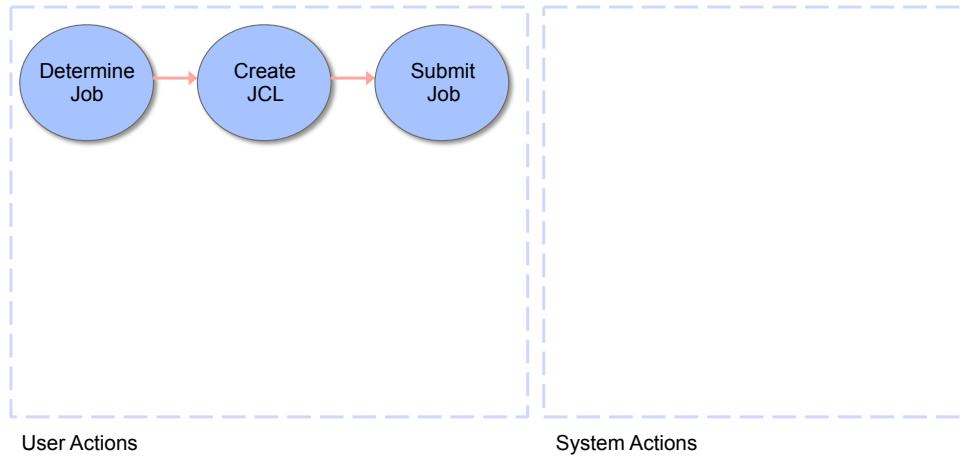
Mainframe jobs operate in batch running the programs or tasks in a series of job steps. Each step usually depends on the previous step for output or results in order to execute.

Mainframe jobs are defined in JCL (Job Control Language) allowing the programmer to define which steps or programs to execute and the conditional processing to handle the results (i.e. job step failed, retry or execute an alternate path of job steps).

For example a common scenario in mainframes is the EOD (End of Day) processing where a series of steps are executed to complete the workload from the day.

These steps may include shutting down access to a particular program used by users during the day, to prevent any further data entry, then the data may be organized in a manner required for the EOD processing (i.e. in sequential order for faster processing) then a series of programs are executed against the data to finalize it (i.e. calculated interest earned, log entries into another database, notify of any exceptions).

Mainframe Job



Determine Job

Define the task that you wish to complete and map out the details and flows of the workload.

Create JCL

Write the job in Job Control Language to implement the job and tasks to be executed.

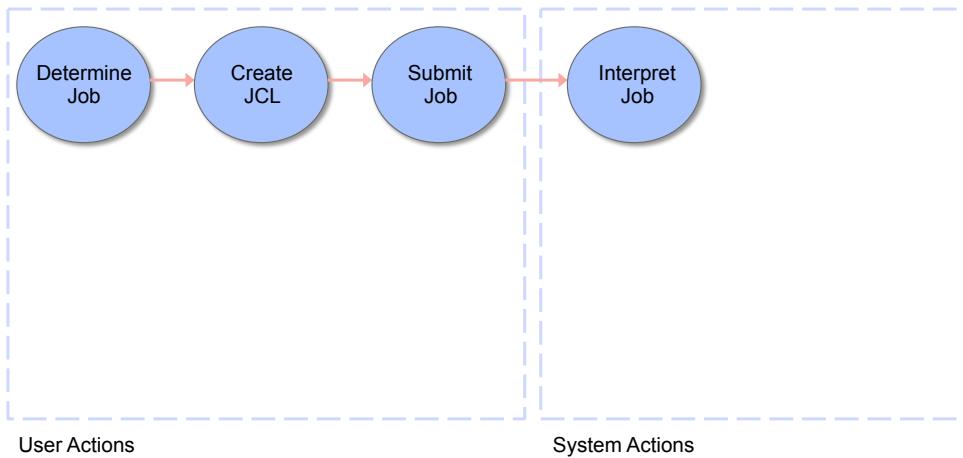
Submit Job

The job is submitted then selected for execution.

Causes JES2 or JES3 to read the job stream from the DASD file and copy it to a job queue or JES spool.

Internal reader - the JES component that processes the input job stream.

Mainframe Job



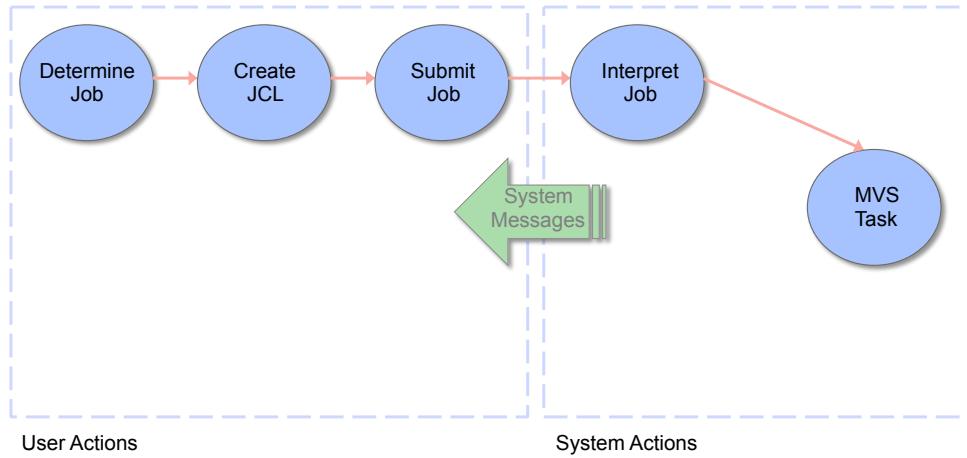
Interpret Job

JES interprets JCL and passes it to MVS. MVS doesn't process jobs in the order they are submitted. JES examines the jobs in the job queue and selects the most important job for execution. JES uses 2 characteristics to classify a job's importance:

job class

priority

Mainframe Job



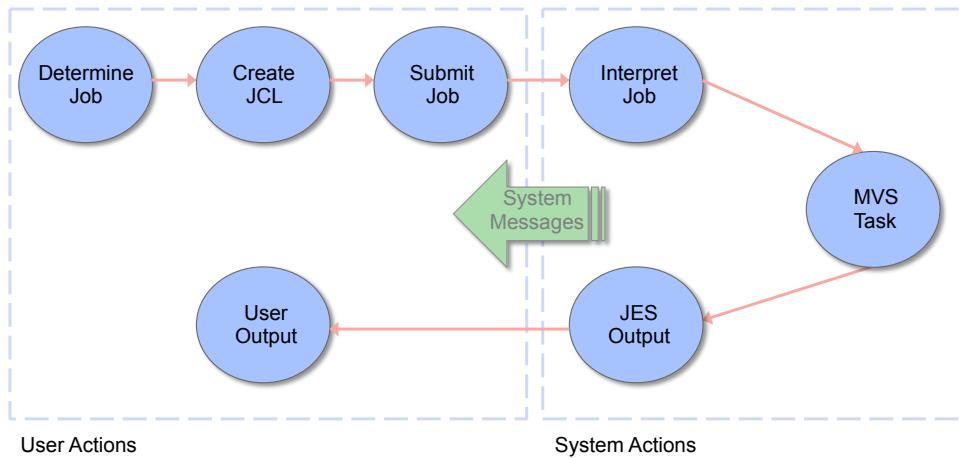
MVS Task

MVS does the work required in the job. Invokes allocation routine that analyze the job to see what resources (units, volumes, data sets) the job needs.

The initiator builds a user region where the user's program can execute.

When the program is complete, the initiator invokes un-allocation routines that release any resources used by the job step.

Mainframe Job



JES Output

JES collects the output and information about the job. SYSOUT data is assigned an output class that determines how the output will be handled.

All outputs of same class are gathered together and printed as a unit.

After a job's output has been processed, the job is purged from the system.

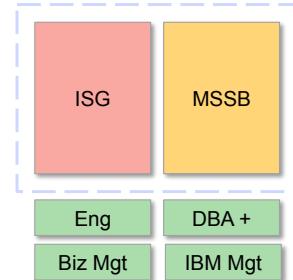
The space the job used is freed for other job's use.

User Views Output

User views and interprets output.

Morgan Stanley Mainframe Environment

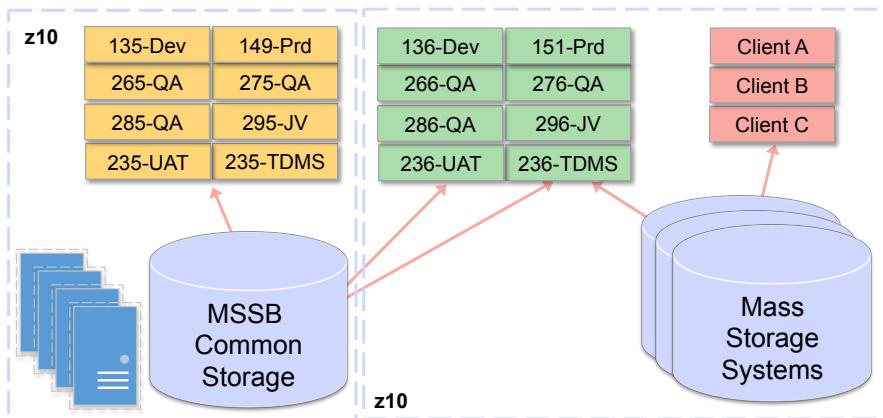
- Mainframe Computing covers two divisions:
 - ISG
 - MSSB
- Mainframe Computing team comprises of four groups:
 - engineering
 - DBA Plus
 - business / Risk Management
 - IBM Relationship Management



Mainframe Computing (MC) integrates Morgan Stanley's ISG mainframe services, supporting all aspects of the trade lifecycle as well as the Firm's books- and records-keeping, with those of MSSB, its wealth management business. MC is comprised of four major groups - Engineering, DBA Plus, Business/Risk Management, and IBM Relationship Management - to create one supporting structure.

MSSB Mainframe Environment

- Implemented in IBM's data centers:
 - Poughkeepsie
 - New York
 - Columbus
- Public / Private Utility Mainframe environment



MSSB operates on a dedicated Morgan Stanley mainframe and a shared IBM On Demand Services public utility mainframe environment.

Private Dedicated Utility / Cloud

8 MSSB LPARS

Target ISV software + dev tools

Public Shared Utility / Cloud

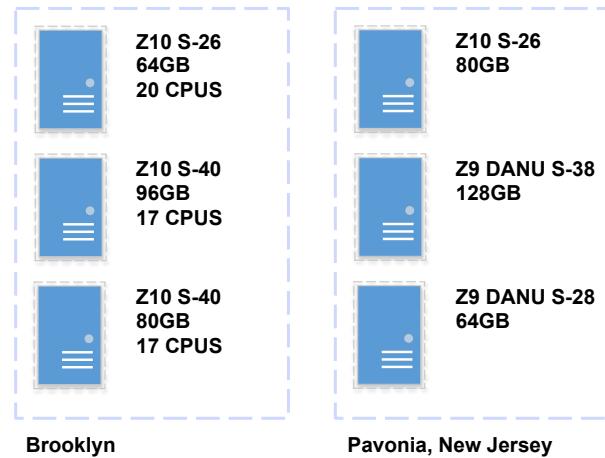
8 MSSB LPARS

IBM software + minimal ISV

Shared CPU with other clients

ISG Mainframe Environment

- Implemented in MS's data centers:
 - Brooklyn
 - New Jersey
- Running z/OS
- 1000 MIPS
- Fully redundant



Created in the early 1980's, the Transaction Analysis Processing System (TAPS) quickly became the mainframe backbone on which Morgan Stanley relies for its trades processing.

In February 2000, TAPSCO was established to unite the TAPS mainframe with its supporting units, producing an integrated "plant" with the mainframe muscle to power the Firm's processing needs now and into the future.

TAPSCO currently consists of online and batch components covering all aspects of the trade lifecycle as well as the Firm's books- and records-keeping.

It is TAPSCO's goal to provide the Firm with increased levels of stability, resiliency, capacity & efficiency through client alignment, innovation and operational excellence.

ISG Mainframe Storage

- EMC
 - production data
 - replication to Newport and Heathrow
 - three copies of production data (BK, NP, HT)
- Hitachi
 - production data
 - replication to Newport and Heathrow
 - three copies of production data (BK, NP, HT)
- IBM
 - test, development and production data
 - data warehousing production data
 - replication to Newport
 - two copies of data (BK, NP)



Within Morgan Stanley there are a number disk arrays available for mainframe processing. The multiplicity of storage options satisfies a few requirements i.e. diversity of vendor, allows for multiple backups of all production data to ensure resiliency and continuity.

Storage can also be tiered based on its underlying performance metrics so that mainframe jobs and processes can take advantage of faster storage if needed.

Storage is also split according to environments i.e. production, test and development and managed accordingly to its importance.

ISG Mainframe Tape

- Real

- Storage Tek is vendor of choice
- 50% of tapes stored in tape library
- 50% of tapes stored in Storage Tek's LSM
- LSM contains 5000 slots for tapes and 30 tape drives
- robotic arm recalls tapes and mounts them
- pass thru ports allow multiple LSM to connect and share tapes

- Virtual

- write to front end disk cache system first (2.4 TB)
- offload to tape drives after batch cycle completes
- 96 VTD's tape drives online to ISG
- speeds batch throughput
- eliminates batch problems associated with tape usage
- i.e. media errors, drive errors, waiting for drive

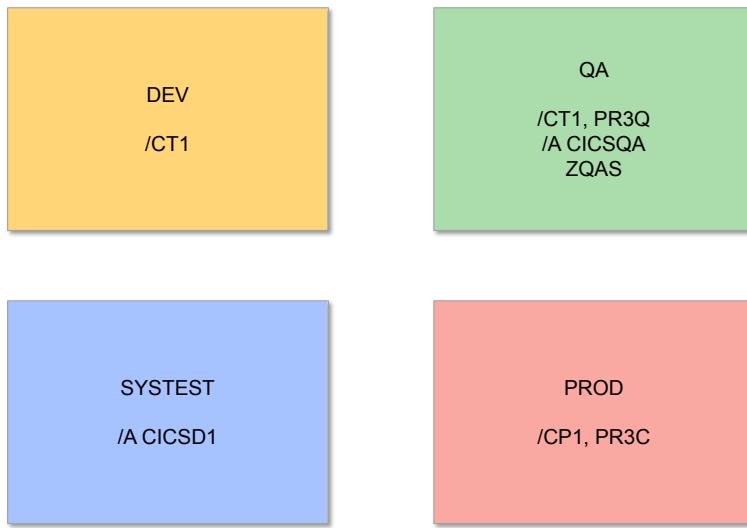


The Morgan Stanley mainframe environment also utilizes tape storage in addition to direct attached storage.

Tapes can store many terabytes worth of data in the Morgan Stanley and can be used for backups and sequential data processing such as batch processing.

The tape library has a front end interface utilizing a disk cache to help with data buffering. In some cases it may take some time for a tape to be loaded into a slot, thus losing valuable processing cycles, but the disk starts accepting the data and then streams it to the tape device when it reports a ready state.

Mainframe Environments



Programs find their way into production via the turnover system called ICUA.

ICUA is one of the in-house change management tools that executes production mainframe turnover requests. For all mainframe applications turnovers ICUA must be used.

TCM usage is required for all ICUA turnovers. To enable TCM usage all mnemonics must be linked to the appropriate GRNs in TAI. ICNs with no approved TCM will not be eligible for Auto Finalization and can only be implemented by liaising directly with the Data Center to get the ICN released to the turnover job. It is the developer's responsibility to gather Post Event TCM approvals for emergency turnovers.

ICUA groups together with associated Natural objects, JCL's, PROC's and Utilib, under an ICN (Implementation Control Number). The turnover request is authorized by a project manager (PM) and is then implemented by a batch process that runs at regular intervals throughout the day.

ICUA allows programmers to group objects that must be implemented together in logical units. JCL, Natural, PROC's and UTILIB's can be included in the same ICN. ICUA's audit trail facilitates querying when a module was last turned over and who performed the turnover.

Z/OS

- 64-bit operating system for IBM mainframes
- Emphasis on backward compatibility
- Derived from OS/390
- Capable of running
 - CICS
 - IMS
 - DB2
 - MQ

z/OS is a 64-bit operating system for mainframe computers, produced by IBM. It derives from and is the successor to OS/390, which in turn followed a string of MVS versions. Like OS/390, z/OS combines a number of formerly separate, related products, some of which are still optional. z/OS offers the attributes of modern operating systems but also retains much of the functionality originating in the 1960s and each subsequent decade that is still found in daily use (backward compatibility is one of z/OS's central design philosophies). z/OS was first introduced in October, 2000.

z/OS supports stable mainframe systems and standards such as CICS, IMS, DB2, RACF, SNA, WebSphere MQ, record-oriented data access methods, REXX, CLIST, SMP/E, JCL, TSO/E, and ISPF, among others. However, z/OS also supports 64-bit Java, C, C++, and UNIX (Single UNIX Specification) APIs and applications through UNIX System Services - The Open Group certifies z/OS as a compliant UNIX operating system - with UNIX/Linux-style hierarchical HFS and zFS file systems. As a result, z/OS hosts a broad range of commercial and open source software of any vintage. z/OS can communicate directly via TCP/IP, including IPv6, and includes standard HTTP servers (one from Lotus, the other Apache-derived) along with other common services such as FTP, NFS, and CIFS/SMB. Another central design philosophy is support for extremely high quality of service (QoS), even within a single operating system instance, although z/OS has built-in support for Parallel Sysplex clustering.

z/OS has a Workload Manager (WLM) and dispatcher which automatically manages numerous concurrently hosted units of work running in separate key-protected address spaces according to dynamically adjustable goals. This capability inherently supports multi-tenancy within a single operating system image. However, modern IBM mainframes also offer two additional levels of virtualization: LPARs and (optionally) z/VM. These new functions within the hardware, z/OS, and z/VM - and Linux and OpenSolaris support - have encouraged development of new applications for mainframes. Many of them utilize the WebSphere Application Server for z/OS middleware.

Because there is only one z/OS version (at least at present), releases are normally called "Release n," though more formally they are "Version 1 Release n" or "V1.n". The IBM Program Number for all z/OS Version 1 releases is 5694-A013

What is ADABAS?

- Very fast OLTP database
- Well suited for high volume data processing
- Based on files instead of tables
- Referential integrity maintained by application code
- Used in conjunction with Natural programming language

ADABAS is an inverted list database. It has been described as "Post-relational" but "Relational Like" in its characteristics. Some differences:

Files, not Tables as the major organizational unit

Records, not Rows as content unit within the organizational unit

Fields, not Columns as components of a content unit

No embedded SQL engine. SQL or another external query mechanism must be provided. SQL Access is provided by the ADABAS SQL Gateway. It provides ODBC, JDBC and OLE DB access to Adabas and enables SQL access Adabas using COBOL programs.

Search facilities may use indexed fields or non indexed fields or both.

Does not natively enforce referential integrity constraints, i.e. parent-child relations must be maintained by application code.

Supports two methods of denormalization: repeating groups in a record ("periodic groups"); and multiple value fields in a record ("multi-value fields").

ADABAS has proven to be very successful in providing efficient access to data and maintaining the integrity of the database. ADABAS is now widely used in applications that require very high volumes of data processing or in high transaction online analytical processing environments.

Natural Language

- Natural – a true 4GL programming language

```
A11 , ,....+....1.....+....2.....+....3.....+....4.....+....5.....+
0010 WRITE 'HELLO WORLD !!'
0020 END
0030
```

- Natural consists of two components
 - programming environment
 - run time environment
- Both environments are in the Natural Nucleus
- Natural is a part interpreted and part machine language
- Natural code is STOWED to generate machine code
 - the nucleus will execute the code

Natural language programming (NLP) is an ontology-assisted way of programming in terms of natural language sentences, e.g. English. A structured document with Content, sections and subsections for explanations of sentences forms a NLP document, which is actually a computer program. An example of natural language programming is in sEnglish that is short for "system English".

The smallest unit of statement in NLP is a sentence. Each sentence is stated in terms of concepts from the underlying ontology, attributes in that ontology and named objects in capital letters. In an NLP text every sentence unambiguously compiles into a procedure call in the underlying high level programming language such as MATLAB, Octave, SciLab, Python, etc.