UNIX and Linux System Administration and Shell Programming

version 22 of September 3, 2012

This book includes material from the http://www.osdata.com/ website and the text book on computer programming.

This is a work in progress. For the most up to date version, visit the website http://www.osdata.com/ and http://www.osdata.com/programming/shell/unixbook.pdf — Please add links from your website or Facebook page.

UNIX and Linux Administration and Shell Programming

chapter 0

This book looks at UNIX (and Linux) shell programming and system administration.

This book covers the basic materials needed for you to understand how to administer your own Linux or UNIX server, as well as how to run your own personal desktop version of Linux or Mac OS X.

This book goes beyond the typical material in a shell scripting class and presents material related to either downloading and compiling existing software (including ports to new hardware and/or operating systems) or for preparing your own software for release via the internet.

requirements

You need a willingness to learn.

You need a working computer or server or access to one.

The computer needs a working version of UNIX, Linux, Mac OS X, AIX, HP/UX, Solaris, etc. (it can be a dual boot computer).

The new version of Mac OS X 10.8.1 (Mountain Lion) is now available on the Mac App Store at www.apple.com as of August 23, 2012. The new version of Mac OS X 10.8 (Mountain Lion) is now available on the Mac App Store at www.apple.com as of July 25th, 2012. Tell them you heard about it from www.osdata.com when you register your new copy.

You need a working connection to the internet, preferably a high speed connection.

You may want to have a domain name of your own and web hosting to try out controlling a server. There is a chapter on how to use GoDaddy to obtain these servcies for low cost with great telephone tech support. The OSdata.com website where this book is offered to the public is hosted by Host Gator. You may use any other hosting service you want.

options

Almost anyone can slog through and learn at least some of this material, but an aptitude for this material greatly helps learning. If you are strong at grammar, then you will probably be able to master. This material. mathematical ability is useful, but not necessary.

Many portions of this book require root or administrator access. While you learn better if you can actually try out each command for yourself, you can just read about root material if you don't have root or administrator access.

Some portions of this book require special software. Most of the software can be downloaded for free. Those with Mac OS X should have the Developer Tools installed. These are available for free on either the install DVD/CD or from Apple at http://connect.apple.com/

A static IP address is in general useful and is required for some portions of this book.

chapter contents

1. cool shell tricks
2. basics of computers
3. UNIX/Linux history
4. choice of shells
5. connecting to a shell (Telnet and SSH; terminal emulator)
6. shell basics (book conventions; root or superuser; starting your shell; login and password; prompt; command example)
7. login/logout (login; select system; account name; password; terminal type; logout; exit)
8. passwd (setting password; local password; periodic changes; 100 most common passwords; secure passwords; superuser)
9. command structure (single command; who; failed command; date; options, switches, or flags; universal time; arguments; options and arguments; operators and special characters)
10. quick tour of shell commands
11. man (using man for help; man sections)
12. cat (creating files; example files for this book; viewing files; combining files)

13. command separator (semicolon)
14. less, more, pg
15. file system basics (graphics examples; directory tree; important directories; home directory; parent and child directories; absolute paths; relative paths; dots, tildes, and slashes)
16. pwd
17. command history
18. built-in commands
19. ls
20. cd
21. cp
22. mv
23. rm (recursive)
24. sysadmin and root/superuser
25. sudo
26. su
27. who
28. major directories
29. shred
30. df
31. du
32. ps
33. w
34. uptime
35. top
36. free
37. vmstat
38. defaults (screencapture; Mac Flashback Trojan)
39. init (init; Linux run levels)
40. ifconfig (view configuration; static IP address)
41. arp
42. netstat (view connections; main info; routing address)
43. route (view connections; routing commands)
44. ping (test packets; measuring)
45. nslookup
46. traceroute (entire route; etiquette)
47. sysstat
48. at (example; removing a job; timing)
49. tar
50. touch (multiple files; specific time)
51. find
52. sed (fixing end of line; adding line numbers)
53. awk (remove duplicate lines)
54. screencapture (from graphic user interface; changing defaults; command line screenshots)
55. installing software from source code
56. test bed

Appendix:

A. computer history
B. Forth-like routines

cool shell tricks for UNIX, Mac OS X, and Linux

chapter 1

summary

This chapter looks at cool shell tricks for UNIX, Linux, and Mac OS X to give you an idea of the power of the shell.

A quick summary of how to get to the shell is included in this chapter (more detailed explanations, including what to do when things go wrong, are in following chapters.

If you need a primer on computer terminology, please look at the next chapter on basics of computers.

cool shell tricks

This chapter has a handful of cool shell tricks. These are intended to show a beginner that a command line shell can be as fun as any graphic user interface and get across the idea that there is a lot of power in the shell that simply doesn't exist in a standard graphic user interface.

definitions

UNIX is one of the ground-breaking operating systems from the early days of computing. Mac OS X is built on top of UNIX. Linux is a variation of UNIX.

The shell is the command line interface for running UNIX (and Mac OS X and Linux) with just typing (no mouse).

operating system The software that provides a computer's basic tasks, such as scheduling tasks, recognizing input from a keyboard, sending output to a display screen or printer, keeping track of files and folders (directories), running applications (programs), and controlling peripherals. Operating systems are explained in more detail for beginners just below.

UNIX UNIX (or Unix) is an interactive multi-user multitasking timesharing operating system found on many types of computers. It was invented in 1969 at AT&T's Bell Labs by a team led by Ken Thompson and Dennis Ritchie. Some versions of UNIX include: AIX, A/UX, BSD, Debian, FreeBSD, GNU, HP-UX, IRIX, Linux, Mac OS X, MINIX, Mint, NetBSD, NEXTSTEP, OpenBSD, OPENSTEP, OSF, POSIX, Red Hat Enterprise, SCO, Solaris, SunOS, System V, Ubuntu, Ultrix, Version 7, and Xenix.

Linux An open-source version of the UNIX operating system.

graphical user interface A graphical user interface (GUI) is a windowing system, with windws, icons, and menus, operated by a mouse, trackball, touch screen, or other pointing device, used for controlling an operating system and application programs (apps). The Macintosh, Windows, Gnome, and KDE are famous examples of graphical user interfaces.

command line interface A command line interface (CLI orcommand line user interface CLUI) is a text only interface, operated by a keyboard, used for controlling an operating system and programs.

shell The shell is the command line interface for UNIX, Linux, and Mac OS X. In addition to intrepetting commands, it is also a programming language.

shell uses

UNIX (and therefore also Mac OS X and Linux) has more than 200 basic commands (also called tools or utilities) that are distributed with the standard operating system. This collection and the ease with which they work together is the major source of the power of UNIX. The vast majority of these standard tools are designed to be used from a command line (the shell).

The shell is most commonly used to control servers. Servers are the computers used to host websites. The most common operating system for the world's web servers is Linux. If you learn shell scripting and system administration, you can run your own server and possibly get a job.

The shell can be used to control a desktop or portable computer. Some tablets and smart phones have a shell. The iPhone actually has a shell, but it can't be accessed witout jailbreaking the iPhone.

The shell will often run even when a computer is partly broken. Both Mac OS X and Linux (as well as almost all versions of UNIX) can be run in a special single user mode. This starts up the computer or server with just the command line shell running. This can be used to service a computer or server, including both diagnosis and repair.

The shell is extremely useful for programming. Even when a programmer uses a graphical integrated development environment (IDE), the programmer is likely to still heavily use the shell for programming support. Some IDEs even have shell access built-in.

command line interface

Before the widespread introduction of graphic user interfaces (GUI), computers were controlled either by punched cards, paper tape, or magnetic tape (a batch system) or a command line interface (CLI) using an interactive terminal (originally, some variation of a teletype machine from the telegraph technology). The earliest computers were controlled by front panel lights and switches or even by directly changing the wiring.

The command line interface on interactive terminals was a major advance. Because of limitations of the early hardware (at a time when a computer's entire memory might be measured in hundreds of bytes), the first CLIs compressed the number of characters, using two or three letter abbreviations for commands and single character switches for options to commands.

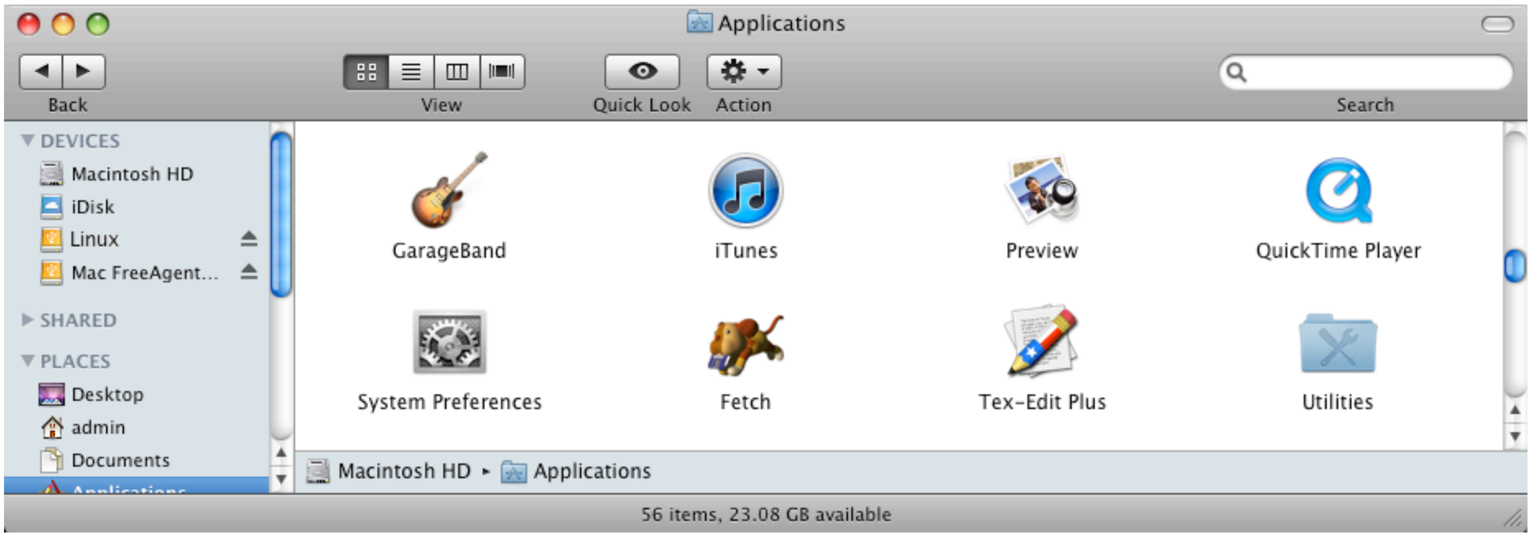
The modern UNIX/Linux shells carry over this early limitation because there is the need to remain backward compatible and still run shell scripts that are decades old, but essential to continued operation of legacy systems.

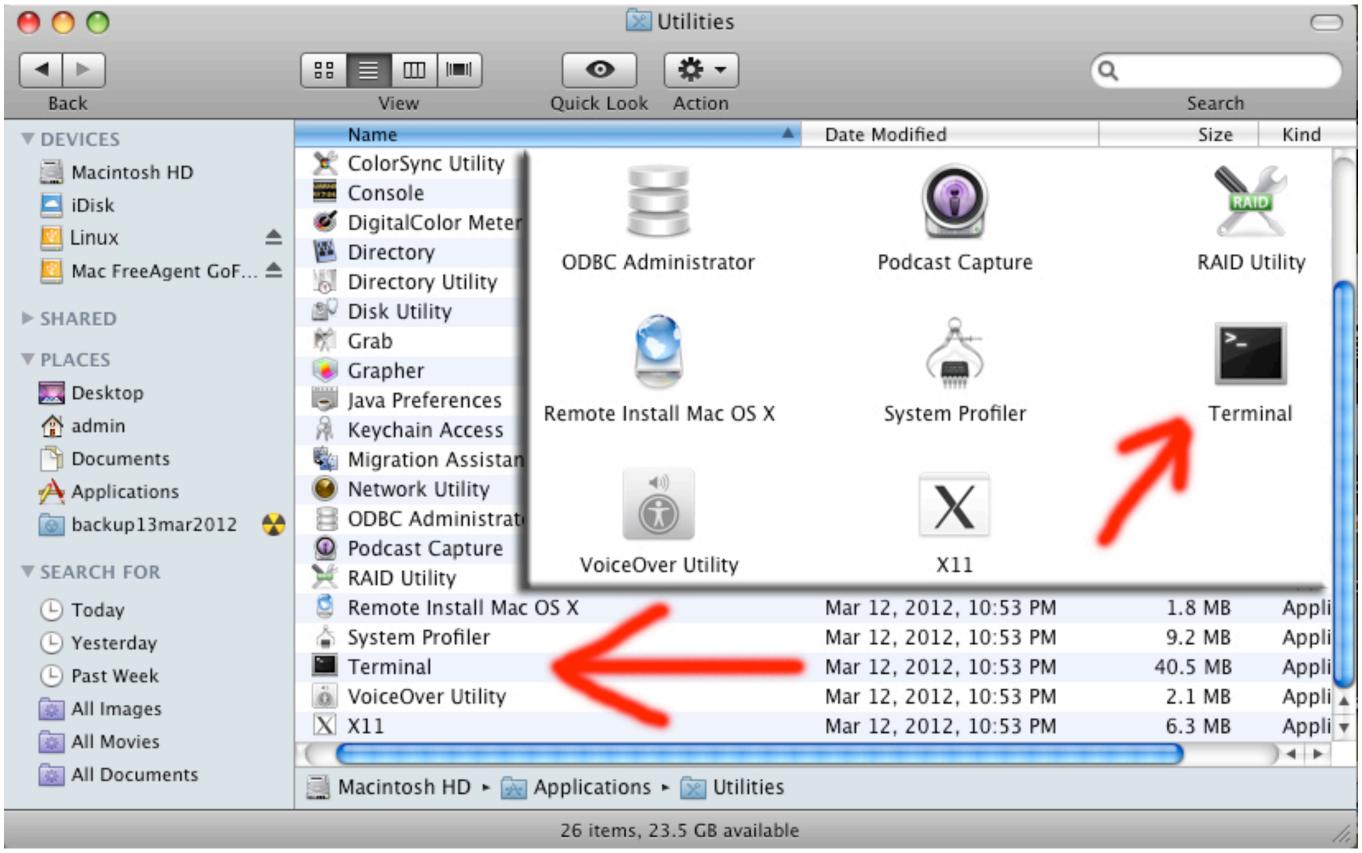
This historical limitation makes it difficult for newcomers to figure out how to use a UNIX/Linux shell.

how to find Terminal

You can run these commands by copying and pasting into Terminal, a program that is available for free and preinstalled on Mac OS X and most versions of Linux. Some of these commands will only work on a particular operating system (this will be indicated), but most can be run from Mac OS X, any distribution Linux, and any kind of UNIX.

On Mac OS X, you will find Terminal by opening the Applications folder (on your main hard drive), then opening the Utilities folder, then scrolling down the list until you find Terminal or Terminal.app. Drag it to your Dock, because you will be using it a lot.



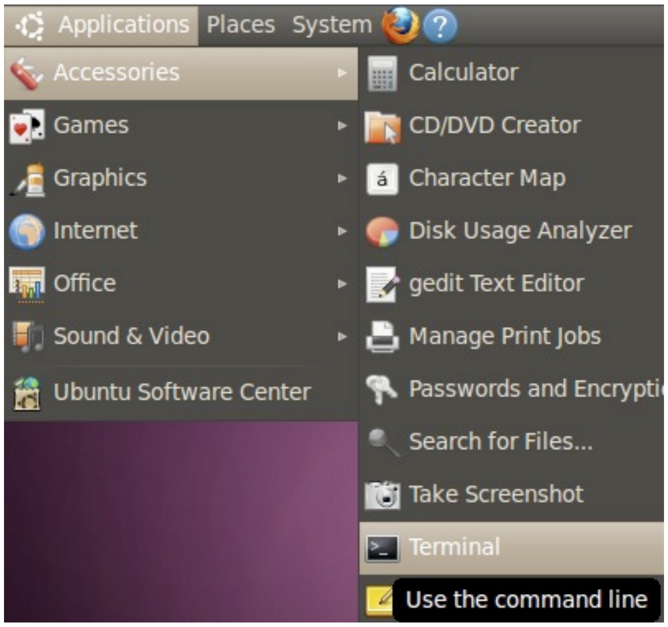


Name	Date Modified	Size	Kind
Remote Install Mac OS X	Mar 12, 2012, 10:53 PM	1.8 MB	Appli
System Profiler	Mar 12, 2012, 10:53 PM	9.2 MB	Appli
Terminal	Mar 12, 2012, 10:53 PM	40.5 MB	Appli
VoiceOver Utility	Mar 12, 2012, 10:53 PM	2.1 MB	Appli
X11	Mar 12, 2012, 10:53 PM	6.3 MB	Appli

On Ubuntu Linux, look in Applications menu > Accessories > Terminal. Single click and hold, then drag to your panel, because you will be using it a lot.

On some versions of Linux, you can press the CONTROL and the ALT and the F1 keys all at once to bring up Terminal.

In Gnome, use the Applications menu > Accessories > Terminal or the keyboard shortcut of CONTROL and ALT and T (all at the same time).



In KDE, use the KMenu > System > Terminal Program (Konsole).

In Linux Mint you can use the keyboard shortcut of holding down the CONTROL and ALT and T keys all at once.

warnings

Be careful about any hints you find on the internet. There are people who suggest very destructive commands in the guise of a useful or fun hint just to trick beginners into destroying their computers.

Be very careful if you use any command that includes `sudo`, because it runs at the root level, allowing complete access to the entire computer with all safeguards turned off. Very powerful. Potentially very distructive in a hurry. There are legitimate hints and cool tricks that use `sudo`, but be careful to type them *exactly* as you you see them in the hint (or copy and paste) and only use `sudo` hints from trusted soruces.

Watch out for anything that includes the command `rm` or `rm *`. That is the remove command versions of it can literally wipe out all of your hard drives in seconds in such a way that only a very expensive data recovery specialist (thousands of dollars) can get your data back.

Also watch out for anything that includes the command `shred`. That is the secure delete and even the most expensive data recovery specialist in the world can’t get your data back.

cool ASCII art

If your computer is connected to the internet, you can use the shell to watch the entire *Star Wars Episode IV* in old fashioned ASCII. Type `telnet towel.blinkenlights.nl` followed by ENTER or RETURN. If you have IPv6, you get extra scenes and color support.

```
$ telnet towel.blinkenlights.nl
```

play a CD

On Linux you can play a CD from the command line. The following example plays the first track of the CD. Change the number to play a different track.

```
$ cdplay play 1
```

On Linux you can get a free coffee cup holder (eject the CD-ROM tray).

```
$ eject
```

basics of computers

chapter 2

summary

This chapter will cover some very basic information on how computers work. I’m not trying to insult anyone’s intelligence. This material addresses some of the questions asked by novice test readers.

major desktop systems

There are only three major desktop computing systems left in common existence: Windows, Macintosh, and Linux.

Apple **Macintosh** was the first commercially successful graphical user interface. That is, a computer where one uses a mouse (or other pointing device) and icons, windows, and menus to control the computer. Since the turn of the century, **Mac OS X** has actually been built on top of UNIX.

UNIX is the last major survivor of the mainframe era (back when computers were so large that they took up an entire room, or even an entire building). Its big advantage of its compeitors was that it is available in free open source versions and it runs on an extremely wide variety of computer hardware (including many computers that are no longer used).

Microsoft **Windows** is loosely based on the Macintosh and for decades dominated the personal computer market.

Linux is a very popular open source variation of UNIX. It is the most common operating system for servers and the third most popular desktop computing operating system. In its early days it was very geeky and very difficult to use, but it now sports two major graphical user interfaces (Gnome and KDE) and is reasonably easy to use.

FreeBSD is the next most popular open source version of UNIX and is commonly used for servers. **Solaris** is the next most popular commercial version of UNIX. IBM and HP both have their own commercial versions of UNIX.

definitions

UNIX is one of the ground-breaking operating systems from the early days of computing. Mac OS X is built on top of UNIX. Linux is a variation of UNIX.

The shell is the command line interface for running UNIX (and Mac OS X and Linux) with just typing (no mouse).

operating system The software that provides a computer’s basic tasks, such as scheduling tasks, recognizing input from a keyboard, sending output to a display screen or printer, keeping track of files and folders (directories), running applications (programs), and controlling peripherals. Operating systems are explained in more detail for beginners just below.

UNIX UNIX (or Unix) is an interactive multi-user multitasking timesharing operating system found on many types of computers. It was invented in 1969 at AT&T’s Bell Labs by a team led by Ken Thompson and Dennis Ritchie. Some versions of UNIX include: AIX, A/UX, BSD, Debian, FreeBSD, GNU, HP-UX, IRIX, Linux, Mac OS X, MINIX, Mint, NetBSD, NEXTSTEP, OpenBSD, OPENSTEP, OSF, POSIX, Red Hat Enterprise, SCO, Solaris, SunOS, System V, Ubuntu, Ultrix, Version 7, and Xenix.

Linux An open-source version of the UNIX operating system.

graphical user interface A graphical user interface (GUI) is a windowing system, with windws, icons, and menus, operated by a mouse, trackball, touch screen, or other pointing device, used for controlling an operating system and application programs (apps). The Macintosh, Windows, Gnome, and KDE are famous examples of graphical user interfaces.

command line interface A command line interface (CLI orcommand line user interface CLUI) is a text only interface, operated by a keyboard, used for controlling an operating system and programs.

shell The shell is the command line interface for UNIX, Linux, and Mac OS X.

the shell

UNIX (and therefore also Mac OS X and Linux) has more than 200 basic commands (also called tools or utilities) that are distributed with the standard operating system. This collection and the ease with which they work together is the major source of the power of UNIX. The vast majority of these standard tools are designed to be used from a command line (the shell).

A shell is primarily a command interpreter.

In a graphical user interface such as Macintosh or Windows, the user controls the computer and programs primarily through pointing and clicking, supplemented by some typing.

In a shell, all of the commands are typed. The shell figures out the meaning of what you typed and then has the computer do as instructed.

But the shell is much more than just a command interpreter. It is also a complete programming language.

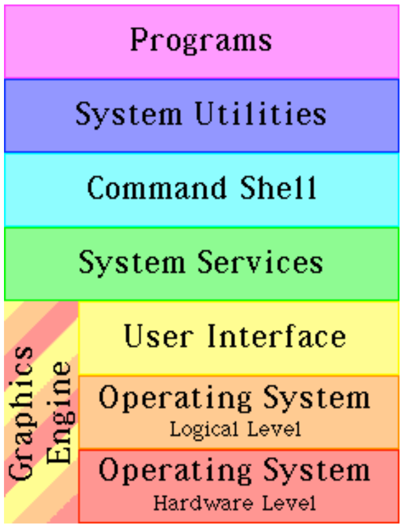
Because the shell is a complete programming language, with sequences, decisions, loops, and functions, it can do things well beyond pointing and clicking. It can take control of your computer and react to changing circumstances.

Programming languages are used to make all of the computer programs and smart phone apps you've ever seen or used. Your imagination is the only limit on the power of the shell. Anything that can be done with a computer can be done with the shell.

operating systems

The seven layers of software are (top to bottom): Programs; System Utilities; Command Shell; System Services; User Interface; Logical Level; and Hardware Level. A Graphics Engine stradles the bottom three layers. Strictly speaking, only the bottom two levels are the operating system, although even technical persons will often refer to any level other than programs as part of the operating system (and Microsoft tried to convince the Justice Department that their web browser application is actually a part of their operating system).

The following are examples of each category:



- Programs: Examples of Programs include your word processor, spreadsheet, graphics programs, music software, games, etc.
- System Utilities: Examples of System Utilities include file copy, hard drive repair, and similar items. On the Macintosh, all the Desk Accessories (calculator, key caps, etc.) and all of the Control Panels are examples of System Utilities.
- Command Shell: The Command Shell on the Macintosh is the Finder and was the first commercially available graphic command shell. On Windows, the Command Shell is a poorly integrated comination of the File Manager and the Program Manager. The command line (C:\ prompt) of MS-DOS or Bourne Shell of UNIX are examples of the older style text-based command shells.
- System Services: Examples of System Services are built-in data base query languages on mainframes or the QuickTime media layer of the Macintosh.
- User Interface: Until the Macintosh introduced Alan Kay's (inventer of the personal computer, graphic user interfaces, object oriented programming, and software agents) ground breaking ideas on human-computer interfaces, operating systems didn't include support for user interfaces (other than simple text-based shells). The Macintosh user interface is called the Macintosh ToolBox and provides the windows, menus, alert boxes, dialog boxes, scroll bars, buttons, controls, and other user interface elements shared by almost all programs.
- Logical Level of Operating System: The Logical Level of the operating system provides high level functions, such as file management, internet and networking facilities, etc.
- Hardware Level of Operating System: The Hardware Level of the operating system controls the use of physical system resources, such as the memory manager, process manager, disk drivers, etc.
- Graphics Engine: The Graphics Engine includes elements at all three of the lowest levels, from physically displaying things on the monitor to providing high level graphics routines such as fonts and animated sprites.

Human users normally interact with the operating system indirectly, through various programs (application and system) and command shells (text, graphic, etc.), The operating system provides programs with services thrrough system programs and Application Program Interfaces (APIs).

basics of computer hardware

A **computer** is a programmable machine (or more precisely, a programmable sequential state machine). There are two basic kinds of computers: analog and digital.

Analog computers are analog devices. That is, they have continuous states rather than discrete numbered states. An analog computer can represent fractional or irrational values exactly, with no round-off. Analog computers are almost never used outside of experimental settings.

A **digital computer** is a programmable clocked sequential state machine. A digital computer uses discrete states. A binary digital computer uses two discrete states, such as positive/negative, high/low, on/off, used to represent the binary digits zero and one.

The French word **ordinateur**, meaning that which puts things in order, is a good description of the most common functionality of computers.

what are computers used for?

Computers are used for a wide variety of purposes.

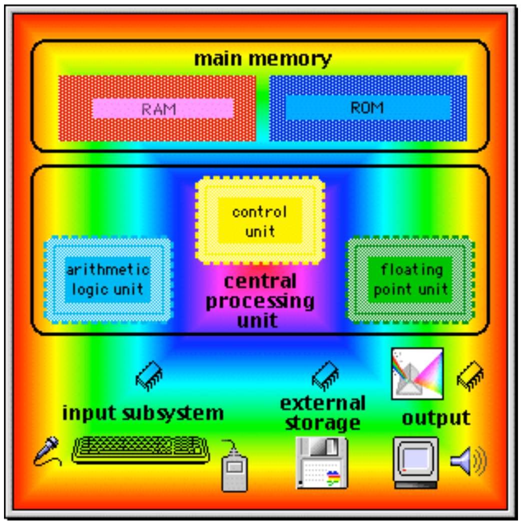
Data processing is commercial and financial work. This includes such things as billing, shipping and receiving, inventory control, and similar business related functions, as well as the "electronic office".

Scientific processing is using a computer to support science. This can be as simple as gathering and analyzing raw data and as complex as modelling natural phenomenon (weather and climate models, thermodynamics, nuclear engineering, etc.).

Multimedia includes **content creation** (composing music, performing music, recording music, editing film and video, special effects, animation, illustration, laying out print materials, etc.) and multimedia playback (games, DVDs, instructional materials, etc.).

Servers includes web servers. Every website is hosted on a computer called a server. When you connect to a website in your web browser, your computer connects to a web server that provides your web browser with all of the parts (text, pictures, Flash, style sheets, JavaScripts, etc.) needed for your web browser to display any particular web page.

parts of a computer



The classic crude oversimplication of a computer is that it contains three elements: processor unit, memory, and I/O (input/output). The borders between those three terms are highly ambigious, non-contiguous, and erratically shifting.

A slightly less crude oversimplification divides a computer into five elements: arithmetic and logic subsystem, control subsystem, main storage, input subsystem, and output subsystem.

- processor
- arithmetic and logic
- control
- main storage
- external storage
- input/output overview
- input
- output

processor

The **processor** is the part of the computer that actually does the computations. This is sometimes called an **MPU** (for main processor unit) or **CPU** (for central processing unit or central processor unit).

A processor typically contains an arithmetic/logic unit (**ALU**), control unit (including processor flags, flag register, or status register), internal buses, and sometimes special function units (the most common special function unit being a floating point unit for floating point arithmetic).

Some computers have more than one processor. This is called **multi-processing**.

The major kinds of digital processors are: CISC, RISC, DSP, and hybrid.

CISC stands for Complex Instruction Set Computer. Mainframe computers and minicomputers were CISC processors, with manufacturers competing to offer the most useful instruction sets. Many of the first two generations of microprocessors were also CISC.

RISC stands for Reduced Instruction Set Computer. RISC came about as a result of academic research that showed that a small well designed instruction set running compiled programs at high speed could perform more computing work than a CISC running the same programs (although very expensive hand optimized assembly language favored CISC).

DSP stands for Digital Signal Processing. DSP is used primarily in dedicated devices, such as MODEMs, digital cameras, graphics cards, and other specialty devices.

Hybrid processors combine elements of two or three of the major classes of processors.

arithmetic and logic

An arithmetic/logic unit (**ALU**) performs integer arithmetic and logic operations. It also performs shift and rotate operations and other specialized operations. Usually floating point arithmetic is performed by a dedicated floating point unit (**FPU**), which may be implemented as a co-processor.

An arithmetic/logic unit (**ALU**) performs integer arithmetic and logic operations. It also performs shift and rotate operations and other specialized operations. Usually floating point arithmetic is performed by a dedicated floating point unit (**FPU**), which may be implemented as a co-processor.

control

Control units are in charge of the computer. Control units fetch and decode machine instructions. Control units may also control some external devices.

A **bus** is a set (group) of parallel lines that information (data, addresses, instructions, and other information) travels on inside a computer. Information travels on buses as a series of electrical pulses, each pulse representing a one bit or a zero bit (there are trinary, or three-state, buses, but they are rare). An **internal bus** is a bus inside the processor, moving data, addresses, instructions, and other information between registers and other internal components or units. An **external bus** is a bus outside of the processor (but inside the computer), moving data, addresses, and other information between major components (including cards) inside the computer. Some common kinds of buses are the system bus, a data bus, an address bus, a cache bus, a memory bus, and an I/O bus.

main storage

Main storage is also called **memory** or internal memory (to distinguish from external memory, such as hard drives).

RAM is Random Access Memory, and is the basic kind of internal memory. RAM is called “random access” because the processor or computer can access *any* location in memory (as contrasted with sequential access devices, which must be accessed in order). RAM has been made from reed relays, transistors, integrated circuits, magnetic core, or anything that can hold and store binary values (one/zero, plus/minus, open/close, positive/negative, high/low, etc.). Most modern RAM is made from integrated circuits. At one time the most common kind of memory in mainframes was magnetic core, so many older programmers will refer to main memory as **core memory** even when the RAM is made from more modern technology. **Static RAM** is called static because it will continue to hold and store information even when power is removed. Magnetic core and reed relays are examples of static memory. **Dynamic RAM** is called dynamic because it loses all data when power is removed. Transistors and integrated circuits are examples of dynamic memory. It is possible to have battery back up for devices that are normally dynamic to turn them into static memory.

ROM is Read Only Memory (it is also random access, but only for reads). ROM is typically used to store thigns that will never change for the life of the computer, such as low level portions of an operating system. Some processors (or variations within processor families) might have RAM and/or ROM built into the same chip as the processor (normally used for processors used in standalone devices, such as arcade video games, ATMs, microwave ovens, car ignition systems, etc.). **EPROM** is Erasable Programmable Read Only Memory, a special kind of ROM that can be erased and reprogrammed with specialized equipment (but not by the processor it is connected to). EPROMs allow makers of industrial devices (and other similar equipment) to have the benefits of ROM, yet also allow for updating or upgrading the software without having to buy new ROM and throw out the old (the EPROMs are collected, erased and rewritten centrally, then placed back into the machines).

Registers and **flags** are a special kind of memory that exists inside a processor. Typically a processor will have several internal registers that are much faster than main memory. These registers usually have specialized capabilities for arithmetic, logic, and other operations. Registers are usually fairly small (8, 16, 32, or 64 bits for integer data, address, and control registers; 32, 64, 96, or 128 bits for floating point registers). Some processors separate integer data and address registers, while other processors have general purpose registers that can be used for both data and address purposes. A processor will typically have one to 32 data or general purpose registers (processors with separate data and address registers typically split the register set in half). Many processors have special floating point registers (and some processors have general purpose registers that can be used for either integer or floating point arithmetic). Flags are single bit memory used for testing, comparison, and conditional operations (especially conditional branching).

external storage

External storage (also called **auxillary storage**) is any storage other than main memory. In modern times this is mostly hard drives and removeable media (such as floppy disks, Zip disks, DVDs, CDs, other optical media, etc.). With the advent of USB and FireWire hard drives, the line between permanent hard drives and removeable media is blurred. Other kinds of external storage include tape drives, drum drives, paper tape, and punched cards. Random access or indexed access devices (such as hard drives, removeable media, and drum drives) provide an extension of memory (although usually accessed through logical file systems). Sequential access devices (such as tape drives, paper tape punch/readers, or dumb terminals) provide for off-line storage of large amounts of information (or back ups of data) and are often called I/O devices (for input/output).

input/output overview

Most external devices are capable of both input and output (I/O). Some devices are inherently input-only (also called read-only) or inherently output-only (also called write-only). Regardless of whether a device is I/O, read-only, or write-only, external devices can be classified as block or character devices.

A **character** device is one that inputs or outputs data in a stream of characters, bytes, or bits. Character devices can further be classified as serial or parallel. Examples of character devices include printers, keyboards, and mice.

A **serial** device streams data as a series of bits, moving data one bit at a time. Examples of serial devices include printers and MODEMs.

A **parallel** device streams data in a small group of bits simultaneously. Usually the group is a single eight-bit byte (or possibly seven or nine bits, with the possibility of various control or parity bits included in the data stream). Each group usually corresponds to a single character of data. Rarely there will be a larger group of bits (word, longword, doubleword, etc.). The most common parallel device is a printer (although most modern printers have both a serial and a parallel connection, allowing greater connection flexibility).

A **block** device moves large blocks of data at once. This may be physically implemented as a serial or parallel stream of data, but the entire block gets transferred as single packet of data. Most block devices are random access (that is, information can be read or written from blocks anywhere on the device). Examples of random access block devices include hard disks, floppy disks, and drum drives. Examples of sequential access block devcies include magnetic tape drives and high speed paper tape readers.

input

Input devices are devices that bring information into a computer.

Pure input devices include such things as punched card readers, paper tape readers, keyboards, mice, drawing tablets, touchpads, trackballs, and game controllers.

Devices that have an input component include magnetic tape drives, touchscreens, and dumb terminals.

output

Output devices are devices that bring information out of a computer.

Pure output devices include such things as card punches, paper tape punches, LED displays (for light emitting diodes), monitors, printers, and pen plotters.

Devices that have an output component include magnetic tape drives, combination paper tape reader/punches, teletypes, and dumb terminals.

UNIX and Linux history

chapter 3

summary

This chapter looks at the history of UNIX and Linux.

The history can help you understand why things are they way they are in UNIX and Linux.

Of particular concern are:

- the small storage space of early computers which resulted in short command names and one character options
- the switchover from teletype I/O to video terminal I/O
- the use of eight bit bytes, ASCII characters, and plain text for interprocess communication
- treating files as collections of bytes and devices, directories, and certain kinds of inter-process communications as files
- the use of small, single-purpose programs that can be easily combined together to perform complex tasks

UNIX history

When UNIX came into existence digital computers had been in commercial use for more than a decade.

Mainframe computers were just starting to be replaced by minicomputers. The IBM 360 mainframe computer and clones by other manufacturers was still the dominant computer. The VAX, with its virtual memory addressing, was still in development. Microcomputers were just an experimental idea. The CDC mainframe, a forerunner of the more famous Cray supercomputers, was still the world's fastest computer.

COBOL was the most common programming language for business purposes. FORTRAN was the most popular programming language for scientific and systems programming. PL/I was a common third language. ALGOL was dominate in the academic community.

Most data entry was still performed using punched cards. Directly connected terminals were typically teletypes (TTY), a technology originally developed for telegraph systems. New terminals combining a keyboard and a monitor were still rare. These were also called CRTs (for Cathode Ray Tube) or dumb terminals or smart terminals (depending on the capabilities). Unlike modern black letters on white background terminals (pioneered with the Apple Lisa and Apple Macintosh) and full color monitors (popularized with the Atari and Commodre Amiga personal computers), these monitors were a single color phosphor (usually green) on a dark gray background. Even though disk drives existed, they were expensive and had small capacities. Magnetic tape was still the most commonly used form of storage of large data sets. Minicomputers were making use of punched paper tape.

Interactive computing was still rare.

In the 1960s, the Massachusetts Institute of Technology (MIT), AT&T Bell Labs, and General Electric attempted to create an experimental operating system called Multics for the GE-645 mainframe computer. AT&T intended to offer subscription-based computing services over the phone lines, an idea similar to the modern cloud approach to computing.

While the Multics project had many innovations that went on to become standard approaches for operating systems, the project was too complex for the time.

Bell Labs pulled out of the Multics project.

UNIX was created at AT&T's Bell Labs in 1969 by a group that included Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, Michael Lesk and Joe Ossanna.

This group of researchers, the last of the AT&T employees involved in Multics, decided to attempt the goals of Multics on a much more simple scale.

"What we wanted to preserve was not just a good environment in which to do programming, but a system around which a fellowship could form. We knew from experience that the essence of communal computing, as supplied by remote-access, time-shared machines, is not just to type programs into a terminal instead of a keypunch, but to encourage close communication," according to Dennis Ritchie.

UNIX was originally intended as a programmer's workbench. The original version was a single-user system. As UNIX spread through the academic community, more general purpose tools were added, turning UNIX into a general purpose operating system.

While Ken Thompson still had access to the Multics environment, he wrote simulations on Multics for UNIX's file and paging system.

Ken Thompson ported the Space Travel computer game from Multics to a Digital Equipment Corporation (DEC) PDP-7 he found at Bell Labs.

Ken Thompson and Dennis Ritchie led a team of Bell Labs researchers (the team included Rudd Canaday) working on the PDP-7 to develop a hierarchial file system, computer processes, device files, a command-line interpreter, and a few small utility programs.

This first UNIX command shell, called the Thompson shell and abbreviated `sh`, was written by Ken Thompson at AT&T's Bell Labs, was much more simple than the famous UNIX shells that came along later. The Thompson shell was distributed with Versions 1 through 6 of UNIX, from 1971 to 1975.

In 1970 Dennis Ritchie and Ken Thompson traded the promise to add text processing capabilities to UNIX for the use of a Digital Equipment Corporation (DEC) PDP-11/20. The initial version of UNIX, a text editor, and a text formatting program called `roff` were all written in PDP-11/20 assembly language.

Soon afterwards `roff` evolved into `troff` with full typesetting capability. The *UNIX Programmer's Manual* was published on November 3, 1971. The first commercial UNIX system was installed in early 1962 at the New York Telephone Co. Systems Development Center under the direction of Dan Gielan. Neil Groundwater build an Operational Support System in PDP-11/20 assembly language.

In 1972 work started on converting UNIX to C. UNIX was originally written in assembly language, but by 1973 it had been almost completely converted to the C language. At the time it was common belief that operating systems must be written in assembly in order to perform at reasonable speeds. Writing UNIX in C allowed for easy portability to new hardware, which in turn led to the UNIX operating system being used on a wide variety of computers.

The PWB shell or Mashey shell, abbreviated `sh`, was a variation of the Thompson shell that had been augmented by John Mashey and others at Bell Labs. The Mashey shell was distributed with the Programmer's Workbench UNIX in 1976.

The Bourne shell, created by Stephen Bourne at AT&T's Bell Labs as a scripting language, was released in 1977 as the default shell in the Version 7 Unix release distributed to colleges and universities. It gained popularity with the publication of *The UNIX Programming Environment* by Brian W. Kernighan and Rob Pike. The book was the first commercially published tutorial on shell programming.

In 1977, the University of California, Berkeley, released the Berkeley Software Distribution (BSD) version of UNIX, based on the 6th edition of AT&T UNIX.

The C shell, abbreviated `csh`, was created by Bill Joy, a graduate student at the University of California, Berkeley. With additional work by Michael Ubell, Eric Allman, Mike O'Brien, and Jim Kulp, it was released in the 2BSD release of BSD Unix in 1978.

It became common to use the C shell for everyday interactive computing and to use the Bourne shell for script programming.

The improved C shell, abbreviated `tcsh`, was created by Ken Greer by September 1975 and merged into the C shell in December 1983. Ken Greer based the shell on the TENEX operating system (hence, the "t" in `tsch`). Mike Ellis, Paul Placeway, and Christos Zoulas made major contributions to `tcsh`.

The Korn shell, abbreviated `ksh`, was created by David Korn at AT&T's Bell Labs and announced at USENIX on July 14, 1983. Mike Veach and Pat Sullivan were also early contributors. The Korn shell added C shell features to the Bourne shell.

When the AT&T broke up in 1984 into "Baby Bells" (the regional companies operating local phone service) and the central company (which had the long distance business and Bell Labs), the U.S. government allowed them to start selling computers and computer software. UNIX broke into the System V (sys-five) and Berkeley Standard Distribution (BSD) versions. System V was the for pay version and BSD was the free open source version.

AT&T gave academia a specific deadline to stop using "encumbered code" (that is, any of AT&T's source code anywhere in their versions of UNIX). This led to the development of free open source projects such as FreeBSD, NetBSD, and OpenBSD, as well as commercial operating systems based on the BSD code.

Meanwhile, AT&T developed its own version of UNIX, called System V. Although AT&T eventually sold off UNIX, this also spawned a group of commercial operating systems known as Sys V UNIXes.

UNIX quickly swept through the commercial world, pushing aside almost all proprietary mainframe operating systems. Only IBM's MVS and DEC's OpenVMS survived the UNIX onslaught.

Some of the famous official UNIX versions include Solaris, HP-UX, Sequent, AIX, and Darwin. Darwin is the UNIX kernel for Apple's OS

X, AppleTV, and iOS (used in the iPhone, iPad, and iPod).

The BSD variant started at the University of California, Berkeley, and includes FreeBSD, NetBSD, OpenBSD, and DragonFly BSD.

Most modern versions of UNIX combine ideas and elements from both Sys-V and BSD.

Other UNIX-like operating systems include MINIX and Linux.

In 1986, Maurice J. Bach of AT&T Bell Labs published *The Design of the UNIX Operating System*, which described the System V Release 2 kernel, as well as some new features from release 3 and BSD.

In 1987, Andrew S. Tanenbaum released MINIX, a simplified version of UNIX intended for academic instruction.

`bash`, which stands for Bourne Again SHell, was created by Brian Fox for the Free Software Foundation and first released on June 7, 1989. `bash` combined features from the Bourne shell, the C shell, and the Korn shell. `bash` is now the primary shell in both Linux and Mac OS X.

The Z shell, abbreviated `zsh`, was written by Paul Flastad in 1990 when he was a student at Princton University.

The Linux operating system was first released on September 17, 1991, by Finnish student Linus Torvalds. With the permission of Andrew S. Tanenbaum, Linus Torvalds started work with MINIX. There is no MINIX source code left in Linux.

Linus Torvalds started work on open source Linux as a college student. After Mac OS X, Linux is the most widely used variation of UNIX.

Linux is technically just the kernel (innermost part) of the operating system. The outer layers consist of GNU tools. GNU was started to guarantee a free and open version of UNIX and all of the major tools required to run UNIX.

In 1992 Unix System Laboratories sued Berkeley Software Design, Inc and the Regents of the University of California to try to stop the distribution of BSD UNIX. The case was settled out of court in 1993 after the judge expressed doubt over the validity of USL’s intellectual property.

See the appendix for a more detailed summary of the history of computers.

choice of shells

chapter 4

summary

This chapter looks at the shells available for Linux and UNIX.

This includes an explanation of what a shell is and a list of the popular shells.

what is a shell?

UNIX (and therefore also Mac OS X and Linux) has more than 200 basic commands (also called tools or utilities) that are distributed with the standard operating system. This collection and the ease with which they work together is the major source of the power of UNIX. The vast majority of these standard tools are designed to be used from a command line (the shell).

A shell is primarily a command interpreter.

In a graphical user interface such as Macintosh or Windows, the user controls the computer and programs primarily through pointing and clicking, supplemented by some typing.

In a shell, all of the commands are typed. The shell figures out the meaning of what you typed and then has the computer do as instructed.

But the shell is much more than just a command interpreter. It is also a complete programming language.

Because the shell is a complete programming language, with sequences, decisions, loops, and functions, it can do things well beyond pointing and clicking. It can take control of your computer and react to changing circumstances.

Programming languages are used to make all of the computer programs and smart phone apps you've ever seen or used. Your imagination is the only limit on the power of the shell. Anything that can be done with a computer can be done with the shell.

choice of shells

Most versions of UNIX and Linux offer a choice of shells.

The term shell comes from a concept of UNIX being a series of layers of software that work together rather than one single monolithic operating system program. The command line shell was part of the outer shell of a series of programs that eventually reach down to the kernel or innermost portion of the operating system.

Steve Bourne created the original Bourne shell called sh.

Some other famous shells were C Shell (csh), Korn Shell (ksh), Turbo C Shell (tsch), and the Z Shell (zsh). A shell that combined the major advntages of each of these early shells was the Bourne Again SHell (BASH).

The default shell in Linux and MacOS X is normally BASH (Bourne Again SHell). The following instruction assumes the use of BASH.

It is possible to change the shell you are currently using (especially useful if you need to run a line or script from an alternative shell).

original shell

The original shell, called the Thompson shell and abbreviated `sh` was written by Ken Thompson at AT&T's Bell Labs, was very rudimentary. The Thompson shell was distributed with Versions 1 through 6 of UNIX, from 1971 to 1975.

Note that the same `sh` is also used for other shells, including the Bourne shell.

The PWB shell or Mashey shell, abbreviated `sh`, was a variation of the Thompson shell that had been augmented by John Mashey and others at Bell Labs. The Mashey shell was distributed with the Prorgammer's Workbench UNIX in 1976.

Bourne shell

Stephen Bourne created the Bourne shell, abbreviated `sh`, at AT&T's Bell Labs as a scripting language.

The Bourne shell was released in 1977 as the default shell in the Version 7 Unix release distributed to colleges and universities. It gained popularity with the publication of *The UNIX Programming Environment* by Brian W. Kernighan and Rob Pike. The book was the first commercially published tutorial on shell programming.

The Bourne shell (or a shell that is compatible and will run Bourne shell scripts) is usually located at `/bin/sh`.

You will also see references to the Bourne shell as `bsh`.

C shell

The C shell, abbreviated `csh`, was created by Bill Joy, a graduate student at the University of California, Berkeley. With additional work by Michael Ubell, Eric Allman, Mike O'Brien, and Jim Kulp, it was released in the 2BSD release of BSD Unix in 1978.

The C shell was much easier to use than the Bourne shell, but was unable to perform anything other than the most simple scripts.

It became common to use the C shell for everyday interactive computing and to use the Bourne shell for script programming.

The improved C shell or TENEX C shell, abbreviated `tcsh`, was created by Ken Greer by September 1975 and merged into the C shell in December 1983. Ken Greer based the shell on the TENEX operating system (hence, the "t" in `tsch`). Mike Ellis, Paul Placeway, and Christos Zoulas made major contributions to `tcsh`.

The Hamilton C Shell was written in 1988 by Nicole Hamilton of Hamilton Laboratories for the OS/2 operating system. In 1992, the Hamilton C Shell was released for Windows NY. The OS/2 version was discontinued in 2003. The Windows version, with adaptations for Windows peculiarities, is still in use.

Korn shell

The Korn shell, abbreviated `ksh`, was created by David Korn at AT&T's Bell Labs and announced at USENIX on July 14, 1983. Mike Veach and Pat Sullivan were also early contributors.

The Korn shell added C shell features to the Bourne shell.

BASH shell

`bash`, which stands for Bourne Again SHell, was created by Brian Fox for the Free Software Foundation and first released on June 7, 1989.

`bash` combined features from the Bourne shell, the C shell, and the Korn shell. `bash` is now the primary shell in both Linux and Mac OS X.

Z shell

The Z shell, abbreviated `zsh`, was written by Paul Flastad in 1990 when he was a student at Princton University.

The Z shell combines features of the Bourne Shell, the C shell, and `bash`.

other shells

The POSIX shell, abbrevaited `sh`, (POSIX = Portable System Interface) is based on the Korn shell and is the official IEEE P1003.2 shell. It is the `/bin/sh` shell on some systems.

The Almquist shell, abbreviated `ash`, was written by Keneth Almquist as a replacement for the Bourne shell written for BSD systems. The `sh` in FreeBSD and NetBSD were based on the Almquist shell.

The Debian Almquist shell, abbreviated `dash`, is a modern replacement for the Almquist shell in Debian and Ubuntu Linux. While the default shell in both Debian and Ubuntu Linux are `bash`, `/bin/sh` is a symlink to `ash`.

`ash` and `dash` are minimalist shells that interpret scripts and are not intended for interactive use. Their small size and fast execution time make them good for embedded systems. Because many scripts call for the use of `/bin/sh`, Debian and Ubuntu UNIX gain speed of execution for scripts, while still allowing `bash` as the primary interactive shell.

The Public domain Korn shell, abbreviated `pdksh`, is based on the Korn shell.

The MirBSD Korm shell, abbreviated `mksh`, is based on the OpenBSD version of the Korn shell and the `pdksh` and was released as part of MirOS BSD.

Busybox is a set of tiny utilities, including a version of `ash`, used in embedded systems.

`rc`, written by Tom Duff, was the default shell for AT&T's Bell Labs' Plan 9 and Version 10 UNIX. Plan 9 was an improved operating system based on UNIX, but was not a significant enough improvement to overtake its predecessor. `rc` is available on some modern versions of UNIX and UNIX-like operating systems.

`es` is an `rc`-compatible shell written in the mid-1990s for functional programming.

Perl shell, abbreviated `psh`, is a shell that combines `bash` and the Perl scripting language. it is available for UNIX-like and Windows operating systems.

Friendly interactive shell, abbreviated `fish`, was released in 2005.

`pysh` is a profile in the IPython project that combines a UNIX-style shell with a Python scripting language shell.

`wish` is a windowing shell for Tcl.Tk.

connecting to a shell

chapter 5

This chapter looks at how to connect to a shell.

If you are connecting to a remote computer (such as web server), then you can use `telnet` or `SSH`.

If you connecting to the shell from a modern graphic user interface, then you need to start up a terminal emulator.

If you connecting to a mainframe computer, minicomputer, or the equivalent, you use any terminal connected to the system.

physical terminal

For old style systems where UNIX runs on a mainframe or minicomputer, there will be actual physical terminals that are connected by wires or MODEM to the mainframe or minicomputer. These systems are increasingly rare.

The terminal may already be at the `login` prompt. If not, try pressing the RETURN key several times until the `login` prompt shows up.

Telnet and SSH

`telnet` or `SSH` are the two basic methods for connecting to a remote server.

`telnet` was the original method. There are `telnet` client programs available for most computer systems. Unfortunately, `telnet` performs all communications in plain text, including passwords, which means that anyone malicious along the path between you and your server can easily read the information needed to hack into your system using your account. For this reason, many servers no longer allow `telnet` access.

`SSH` is Secure Shell. It has the same basic capabilities as the older `telnet`, but includes security, including encrypted communications. There are `SSH` client programs for most modern computer systems.

You can also use `SSH` through a terminal emulator, which is how many system administrators now access their servers from their personal or portable computer.

terminal emulator

As mentioned in the history chapter, the UNIX shells were accessed through interactive terminals, originally teletype machines and later special combinations of keyboards and cathode ray tubes (CRTs).

There are programs for graphic user interfaces that allow you to interact with a shell as if you were using one of those ancient terminal devices. These are called terminal emulators (they emulate the signals of a terminal).

Some common terminal emulators include eterm, gnome-terminal, konsole, kvt, nxterm, rxvt, terminal, and xterm.

Note that if you are logging in to a web server or other remote computer, you should use an SSH (Secure SHell) client program instead of a terminal emulator. A terminal emulator program is for gaining shell access to a personal computer or workstation. With a little bit more knowledge, you can use the shell on a personal computer or workstation as your SSH client.

shell basics

chapter 6

summary

This chapter looks at UNIX (and Linux) shell basics.

This includes an explanation of what a shell is, how to get to your shell, and a simple example of how to use your shell.

The short version of this chapter is to start up your terminal emulator (or SSH client), type in your account name at the `login` prompt, followed by your password at the `password` prompt (using the RETURN or ENTER button at the end of each line). Then type a valid command (such as `who`) and an invalid command (such as `asdf`) and see the results. The rest of the commentary helps you through common questions and problems.

book conventions

The following conventions are used in this book.

Examples are displayed in a gray box. In the PDF version of this book the gray box does *not* appear, but the box is indented from the regular text. In the on-line version at www.osdata.com the gray box displays correctly. I need to find someone with a legal copy of Adobe Acrobat to help out. If you are in the Costa Mesa, California, area and can help out with a legal copy of Adobe Acrobat, please contact me.

Additionally, anything that will appear on the screen (whether you type it or the shell produces it as output) will appear `in a monospace type`. This will help identify the examples in the PDF version. This also applies to material contained in ordinary descriptive paragraphs that you will type.

```
$ date
Wed Nov 10 18:08:33 PST 2010
$
```

Bold face type is used to indicate items that you type.

There will be no indication of the correct locations to type RETURN or ENTER or other similar special characters. These normally happen at the end of every input line. There will be reminders in the descriptive text near the beginning of the book, as well as in other descriptive text where there is a need for a special reminder or an unusual convention or an unusual keystroke.

```
$ date
```

Bold italics type is used to indicate items that you must customize, such as file names or user account name. In a descriptive paragraph, these items will be in *ordinary italics*.

```
$ AccountName
```

Material that the shell will output will be in ordinary `monospace type`. In some cases where the material will be variable, the shell output will be in *`italics`*.

```
Wed Nov 10 18:08:33 PST 2010
VariableInformation
$
```

root or superuser

Do **not** run your terminal emulator or shell in root or superuser.

Root or superuser has complete control over the entire computer. A regular user account has limitations.

Experienced programmers run BASH as a regular user to protect themselves from destructive mistakes — and they are already skilled at using BASH.

A beginner running BASH as superuser or root can turn a simple typing mistake into a disaster.

If you already have a regular user account created, make sure you use it. If you only have the superuser or root account running, immediately create a regular user account and switch to it for all of your learning exercises.

starting your shell

Some information you may need includes the name or URL of your server or computer (sometimes called the host), your account name, and your assigned password. if you are working on a remote server (such as a web server), you need to get this information from your hosting company or system administrator. If you are using a large multi-user system, you need to get this information from your system administrator. If you are using a personal computer or workstation, you probably set these for yourself.

If you have just purchased a new personal computer or workstation, ask your salesperson for the defaults or look them up in the manual. Your install CD or DVD is likely to have a utility that can be used to create accounts and set passwords. The Mac OS X install DVD has utilities for both purposes.

If you are the first person to work with a large system or computer, refer to the manuals that came with your computer. You will find information on the root/superuser account. That is a powerful account and should be reserved for special occasions when that kind of power is needed. Look in the manual section with a title similar to "Getting Started" and then look for an account named *user*, *guest*, *tutor*, or similar name that suggests an ordinary user account.

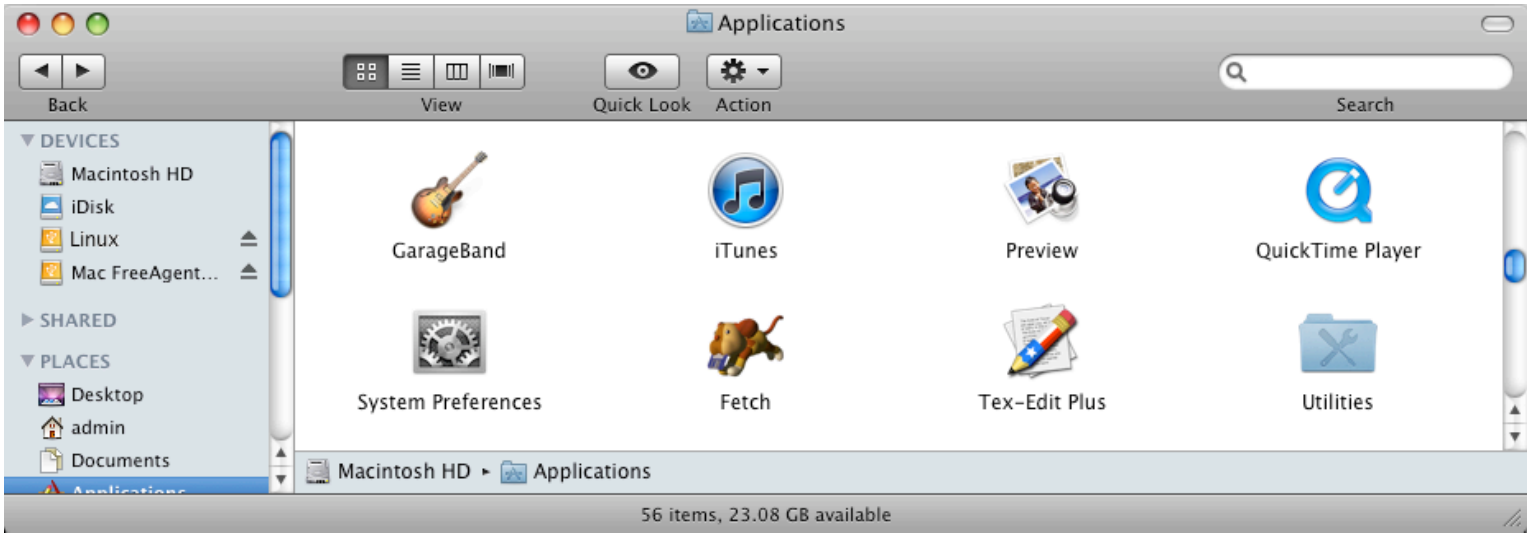
You login remotely into a web server through SHH (Secure SHell). There are many SSH programs available. You do not have to have a matching operating system. You can use an SSH client program on Windows, Macintosh, Linux, or almost any other operating system to sign into a Linux or UNIX server.

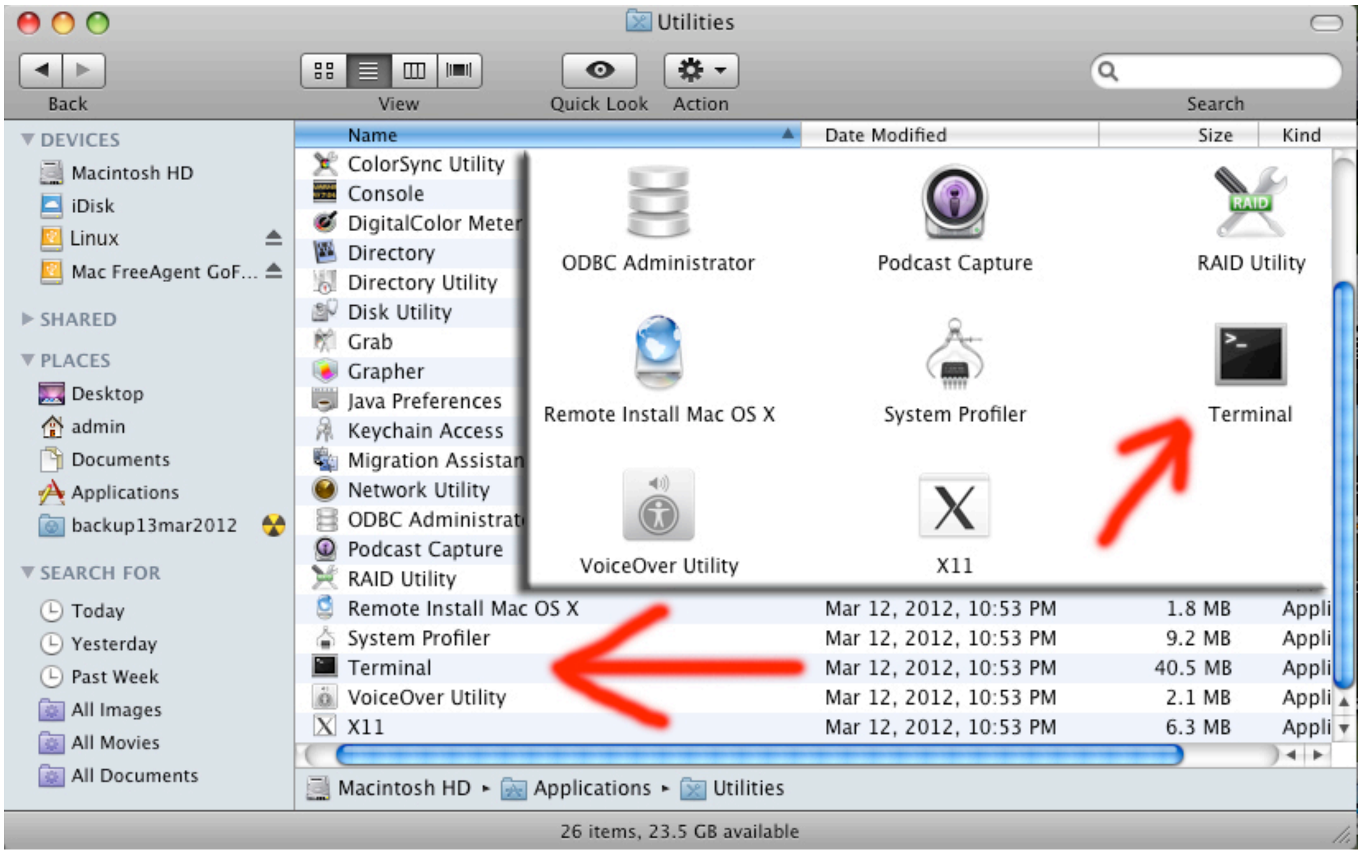
You login into a large multi-user traditional command line UNIX system by typing your user name, ENTER, your password, and ENTER (in that order). This will enter you into the default shell set up for your account.

On a computer that runs a graphic interface (such as Gnome, KDE, and Mac OS X) you will want to use a terminal emulator program. Typically a terminal emulator program starts with you already logged-in to the same account that you have running in the graphic user interface.

On Mac OS X and most versions of Linux, the icon for the Terminal program will look like this:

On a Mac OS X computer, Terminal will be located in the Utilities directory, which is located in the Applications directory. When you find it, you can drag it onto the Dock to make it easily accessible. Apple purposely hid the UNIX command line to avoid confusing the typical consumer.





Name	Date Modified	Size	Kind
Remote Install Mac OS X	Mar 12, 2012, 10:53 PM	1.8 MB	Appli
System Profiler	Mar 12, 2012, 10:53 PM	9.2 MB	Appli
Terminal	Mar 12, 2012, 10:53 PM	40.5 MB	Appli
VoiceOver Utility	Mar 12, 2012, 10:53 PM	2.1 MB	Appli
X11	Mar 12, 2012, 10:53 PM	6.3 MB	Appli

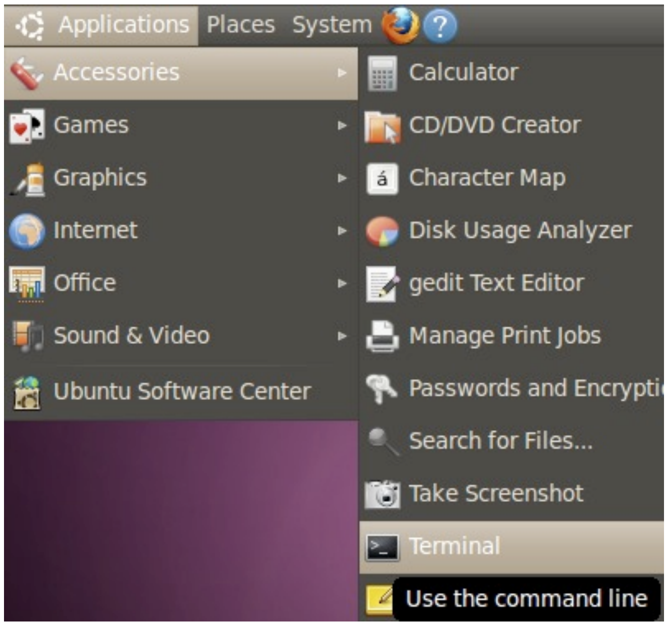
On Linux/KDE, look for Konsole or Terminal in the Utilities menu.

On Linux/Gnome, look for color xterm, regular xterm, or gnome-terminal in the Utilities menu.

On Ubuntu Linux, look in Applications menu > Accessories > Terminal. Single click and hold, then drag to your panel, because you will be using it a lot.

On some versions of Linux, you can press the CONTROL and the ALT and the F1 keys all at once to bring up Terminal.

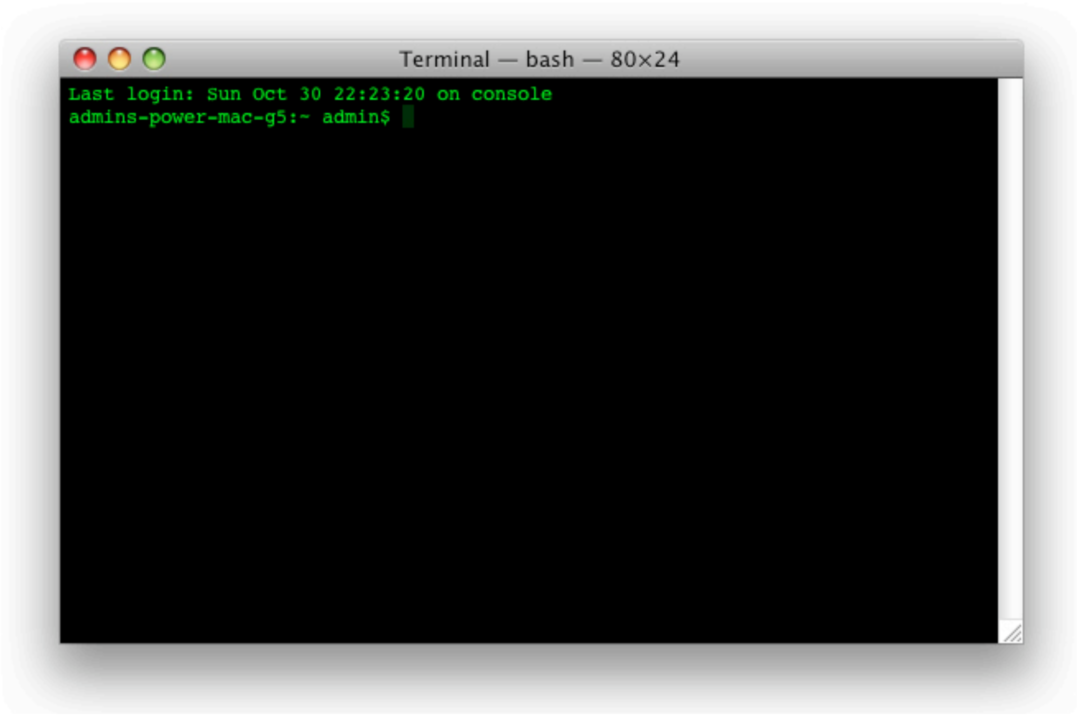
In Gnome, use the Applications menu > Accessories > Terminal or the keyboard shortcut of CONTROL and ALT and T (all at the same time).



In KDE, use the KMenu > System > Terminal Program (Konsole).

In Linux Mint you can use the keyboard shortcut of holding down the CONTROL and ALT and T keys all at once.

You may want to try out the various terminal emulators and decide which works best for you (each offer slightly different features).



```
Last login: Sun Oct 30 22:23:20 on console
admins-power-mac-g5:~ admin$
```

login and password

At the `login` prompt type your account name (in lower case letters) followed by ENTER or RETURN.

```
login: accountname
Password:
```

At the `password` prompt type your password followed by ENTER or RETURN.

```
Password: password
$
```

If you encounter problems, see the next chapter for help.

prompt

Once the shell is running it will present a shell prompt. This shell prompt is normally the U.S. dollar sign ($). Other common prompts are the percent sign (%) and the pound sign (#). The shell prompt lets you know that the shell is ready and waiting for your input.

```
$
```

The most common prompt in the Bourne shell (`sh` or `bsh`) and Bourne Again shell (`bash`) and Korn shell (`ksh`) is the U.S. dollar sign ($).

```
$
```

When the the Bourne Again shell (`bash`) or Korn shell (`ksh`) is running as root or superuser, the prompt is changed to the U.S. number or pound sign (#) as a clear reminder that you are using a much more powerful account.

```
#
```

The Bourne Again shell (`bash`) shell typically reports the current working directory and user name before the `$` prompt, often inside of square braces. An example:

```
[admin:~ admin]$
```

The most common prompt in the C shell (`csh`) and TENEX C shell (`tsch`) is the percent sign (%).

```
%
```

When running as root or superuser, the C shell (`csh`) shows the U.S. number or pound sign sign (#).

```
#
```

The Z shell (`zsh`) shows the short machine name followed by the percent sign (%).

```
mac%
```

When running as root or superuser, the Z shell (`zsh`) shows the short machine name followed by the U.S. number or pound sign sign (#).

```
mac#
```

Almost all of the shell examples in this book will use **bold face** to show what you type and plain fixed space characters to show shell output. Items that you need to modify according to your local needs are normally going to be presented in *italics*.

You are likely to see information before the actual command prompt. It may the current working directory. It may be the current user name. There are ways to change what is presented. For now, don't worry about it, just be aware that you are likely to see some text before the prompt.

example command

You can run a UNIX (or Linux or Mac OS X) command (also called a tool) by typing its name and then the ENTER or RETURN key.

The following example uses the **date** command or tool.

```
$ date
Wed Nov 10 18:08:33 PST 2010
$
```

The format is: day of the week, month, day of the month, 24 hour time, time zone, year.

failed command

BASH will let you know when you have typed something it doesn't understand.

Type "asdf" and then the ENTER or RETURN key. You should see a message similar to the following:

```
$ asdf
-bash: asdf: command not found
$
```

login/logout

chapter 7

summary

This chapter looks at `login` and `logout`, a pair of UNIX (and Linux) commands.

On a personal computer or workstation you probably don't need to `login`. You can just start your terminal emulator (see previous chapter).

On a multi-user UNIX or Linux system, such as a web server, you will need to `login`. This chapter furthers the discussion `login` in more detail than the previous chapter's introduction, including special cases and solutions to common problems.

In particular, this chapter discusses choosing the system and terminal type, rare options that you might encounter.

This chapter also discusses the `logout` command and the importance of always logging out from a running system.

login

`login` is a UNIX command for logging into a UNIX system.

`login` is a builtin command in `csh`. There is also an external utility with the same name and functionality.

In many modern systems, the functionality of login is hidden from the user. The login command runs automatically upon connection. In some old computers you may need to type a special character or even type `login` to bring up the login program.

If you login remotely to a server you will be prompted for your user/account name and password. If you start up a terminal emulator and shell from a graphic user interface you will probably already be authenticated and immediately go to the shell (although some of the startup activity of login will still be done for you).

The activities that login performs at startup of a shell will be covered in a later chapter, because you need more familiarity with the shell before you can understand that material.

Starting a shell is covered in the previous chapter about shell basics.

select system or host

If you are signing into a system that combines multiple computers, you will first need to indicate which computer you are logging into.

You may already be logged in if you are using a terminal emulator on a personal computer or workstation. If so, you don't need this step.

You will see a prompt indicating a choice of computer, such as (only one of these will appear):

```
host:
pad:
request:
system:
```

You will see only one of these (or similar) choices.

If you see `login:` instead, then you don't have to worry about choosing the system.

Enter the identifying name of the computer system you were assigned to use, followed by the ENTER or RETURN key. Your system administrator can provide you with this information.

On some networked systems you can abbreviate with the first two or three characters of the computer name (as long as it is enough to uniquely identify which computer is intended).

If you are successful, you will be presented with the `login:` prompt.

account name

You may already be logged in if you are using a terminal emulator on a personal computer or workstation. If so, you don't need this step.

Ask your system administrator for your account name. If this is your own personal system, then use the account name set up during installation. Many modern install disks include a utility for creating a new account.

At the `login:` prompt, enter your account name, followed by the ENTER or RETURN key.

Type your account name in lower case letters. UNIX assumes that an account name in ALL CAPITAL LETTERS indicates an old input/out device that doesn't support lower case letters (such as an old style teletype, or TTY). Also, if you use upper case letters for the login, then you can not use lower case letters in the password.

If you enter a valid account name, you will be asked for your password. On many computers, you will be asked for a password even if you enter an incorrect password. This is a security measure to help prevent guessing account names.

```
login: accountname
Password:
```

See the chapter on login and logout for more information.

password

You may already be logged in if you are using a terminal emulator on a personal computer or workstation. If so, you don't need this step.

Ask your system administrator for your password. If this is your own personal system, then use the password set up during installation. Many modern install disks include a utility for changing the password for an existing account or for creating a new account with a new password.

At the `Password:` prompt, enter your password, followed by the ENTER or RETURN key.

```
Password: password
$
```

In some cases, a new account doesn't have a password. Also, it is common for Mac OS X to have no password for the user account.

If your account doesn't have a password, just press the ENTER or RETURN key.

Change your password to a secure password at your earliest opportunity. If you don't know how to do this yet, the upcoming chapter on passwd (yes, that is the correct spelling) will tell you how. That same chapter gives advice on how to select a secure password.

Failure will be indicated by an error message. The exact error message varies by system, but will probably be something similar to:

```
Login incorrect
```

The most likely reasons are:

1. Incorrect account name.
2. Incorrect password.
3. You typed the account name in upper case letters (check the CAPS LOCK key).

You should have a new login prompt and can easily start over.

If you logged in in upper case, the shell will display in upper case letters and expect you to type only in upper case letters.

If you accidentally typed the account name in upper case letters, but haven't yet typed in the password, many UNIX systems will display aa backslash character in front of the password prompt.

If you realize you made a mistake on the account name and want to start over, hold down the CONTROL key (often marked CTRL) and the D key at the same time. This will stop the login program and start over again from the beginning.

Note that at one time there was a security hole because of the Control-D. A hacker would start the login, type Control-D, and then try to slip in a command to the shell before a new login program could be started. This security hole no longer exists on shell login, but this same type of timing attack is still used for other security holes.

terminal type

On some older UNIX systems you may be asked to provide the terminal type you are using. Some modern terminal emulator programs offer the choice of emulating various different terminals from the early UNIX era. These different terminal types had different specialized capabilities.

Because the early traditional UNIX was created primarily on Digital Equipment Corporation (DEC) computers (first the PDP, followed by the VAX), the default terminal device was the DEC VT100. Most modern terminal emulator programs act like the VT100.

On older UNIX systems you may see the terminal type prompt:

```
Term:
```

In some cases, the prompt may give the default terminal setting in parenthesis.

```
Term = (VT100)
```

If the default terminal setting is correct, simply press the RETURN or ENTER key.

Otherwise, type in the correct terminal type, followed by the ENTER or RETURN key.

If you are uncertain, try the `vt100` or `UNKNOWN`, followed by ENTER or RETURN.

Some common terminal types:

- h19
- tvi925
- vt100
- wyse50

After you are logged in, you can set the terminal type (capitalization matters).

Shell	Command
`csh` or `tcsh`	`setenv TERM vt100`
`sh`	`TERM=vt100; export TERM`
`ksh`, `bash`, or `zsh`	`export TERM=vt100`
`VMS`	`set term/device=vt100`

If you do not know your shell type, use the following command (followed by ENTER or RETURN):

```
$ echo $SHELL
```

You will need to enter the appropriate command each time you login.

Alternatively, you can add the appropriate command line to the initialization file in your home directory.

Shell	Login file
`csh`	`.cshrc` or `.login`
`tcsh`	`.cshrc`
`ksh`	`.profile`
`zsh`	`.zshrc`
`bash`	`.bash_profile`

VMS users need to modify the login.com file, adding the line `$set term/device=vt100` .

logout

When you finish using the shell, type the `exit` or `logout` command. This will either display a new login prompt or close the terminal emulator window.

When you finish using the shell, type the `exit` or `logout` command. This will either display a new login prompt or close the terminal emulator window. if you are logged in remotely to a server, this will break your connection. If you are logged in physically on a large UNIX system, this will prevent someone else from abusing your account.

Some of the possible variations of this command include: `bye`, `exit`, `lo`, `logout`, and `quit`.

Typing CtrlL-D (holding down the CONTROL key and the D key at the same time) will also log you out of most shells. The Ctrl-D tells the shell it has reached the end-of-file (EOF). Because the shell is a filter and filters terminate when EOF is reached, the shell terminates and you log off the system or return to the parent process.

While it might not be a big deal to skip this step on a single user computer that you never leave unattended, forgetting to logout from a remote server is a serious security hole. If you leave your computer or terminal unattended, someone else can sit down and gain access to all of your files, including the ability to read, modify, or delete them. The imposter can send spam emails from your user ID. The imposter can even use your computer or account to attempt to hack or damage or break into any system in the world from your user ID. If this occurs from a school or business account, you will be responsible for any damage done.

Most modern systems accept either `logout` or `exit`.

On Mac OS X, typing `logout` produces the following result:

```
$ logout

[Process completed]
```

On Mac OS X, typing `exit` produces the following result (notice that the shell automatically runs `logout` for you):

```
$ exit
logout

[Process completed]
```

A successful `logout` will stop all running foreground or background processes. There is a way to keep processes running, which is useful for things like starting up your Apache web server and MySQL data base, but the method will be discussed later.

```
$ logout
There are suspended jobs.
```

If you see the message “There are stopped jobs.” it means that you have one or more suspended jobs. The shell is letting you know in case you forgot about them. If you type `logout` again without running the `jobs` in between, the system will go ahead and log you out and terminate all suspended jobs.

`logout` is a builtin command in `csh`.

exit

`exit` is a built-in shell command (for the shells that support it), while `logout` is a program. In the csh (C Shell), `logout` is an internal built-in command. sh (the original Bourne shell) and ksh (the Korn Shell) do not officially have the `logout` command, but many modern versions support it.

`exit` can be used by itself as a substitute for `logout` (as mentioned above).

`exit` can also be used to send a return value (often an error code) to the parent process. Simply add an integer following `exit`. This is particularly useful in scripting.

```
$ exit n
```

`exit` will be covered later in its own chapter.

passwd

chapter 8

summary

This chapter looks at `passwd`, a UNIX (and Linux) command.

`passwd` is used to change your password.

This chapter also includes the list of the 100 worst (most commonly used) passwords.

As mentioned in the previous chapter, you should change your password from the original default or assigned password to a secure password that nobody else knows. And please don't leave the new password on a sticky note attached to your computer.

setting your password

Type `passwd` followed by the ENTER or RETURN key.

```
        $ passwd
```

You will be prompoted to give your current (old) password (to make sure it is really you) and then prompted to enter your new password twice. For security purposes, the password is typically replaced with asterisks or some other character so that nobody can read your password over your shoulder. To make sure that you have typed what you thought you typed you are asked to type the new password twice. The two copies must match before your new password replaces your old password.

local password

The password set by `passwd` is your *local* password. On a single user system, this is probably your only password.

On Mac OS X the use of the `passwd` may or may not be sufficient to change your password for the entire system. This depends on which version of Mac OS X you are using. It is best to change your password using the install disc. If you do not have a copy of the install disc, there are instructions on the internet on how to manually change the password.

On a large system, there may be multiple passwords spread across multiple computers. The `passwd` command will only change the password on the one server that you are currently logged into (normally through SSH). You may need to use `yppasswd` or a web interface to change your password for the entire system.

You can check for your account or username in `/etc/passwd`. If it's not listed there, then don't use the `passwd`. Check with your system administrator.

periodic changes

Whenever you first login into a new system, the first thing you should do is change your password. In particular, immediately change the initial root password for a new system. Leaving the initial default password is a huge security hole and hackers do try all of the standard default passwords to see if they can find an easy way into a computer. Even with a user account, it is common for initial passwords to be generated poorly and be easy for hackers to guess.

Additionally, you want to change your password on a regular basis. It only takes a few months to figure out a password through brute force attacks. Some systems require that you change your password on a regular basis. Once a month is a good time period. More often if you suspect that someone saw you typing or there is any other possibility that your password might have been compromised.

You can set up your account to remind you to change your password on a regular basis. If you are the system administrator, you can set up these reminders for everyone (and should do so). As system administrator you can even require that users change their passwords on a regular basis (or they become locked out and have to come to you to beg for re-entry). As system administrator you can also set up a system that requires (or even suggests) secure passwords.

100 most common passwords

Always avoid the common passwords. These are the most common passwords as of June 2012:

1. password
2. 123456
3. 12345678
4. 1234
5. qwerty
6. 12345
7. dragon
8. pussy
9. baseball
10. football
11. letmein
12. monkey
13. 696969
14. abc123
15. mustang
16. michael
17. shadow
18. master
19. jennifer
20. 111111
21. 2000
22. jordan
23. superman
24. harley
25. 1234567
26. trustno1
27. iloveyou
28. sunshine
29. ashley
30. bailey
31. passw0rd
32. 123123
33. 654321
34. qazwsx
35. Football
36. seinfeld
37. princess
38. peanut
39. ginger
40. tigger
41. fuckme
42. hunter
43. fuckyou
44. ranger
45. buster
46. thomas
47. robert
48. soccer
49. fuck
50. batman
51. test
52. pass
53. killer
54. hockey
55. babygirl
56. george
57. charlie
58. andrew
59. michelle

60. love
61. jessica
62. asshole
63. 6969
64. pepper
65. lovely
66. daniel
67. access
68. 123456789
69. joshua
70. maggie
71. starwars
72. silver
73. william
74. dallas
75. yankees
76. 666666
77. hello
78. amanda
79. orange
80. biteme
81. freedom
82. computer
83. sexy
84. nicole
85. thunder
86. heather
87. hammer
88. summer
89. corvette
90. taylor
91. fucker
92. austin
93. 1111
94. merlin
95. matthew
96. 121212
97. golfer
98. cheese
99. martin
100. chelsea

Approximately 4.7% of all users have a password of *password*. 8.5% have one of the top two passwords. 9.8% (nearly one tenth) have one of the three top passwords. 14% have one of the top 10 passwords. 40% have one of the top 100 passwords. 79% have one of the top 500 passwords. 91% have one of the top 1,000 passwords.

secure passwords

It is important to have secure passwords.

The more characters, the more secure. A minimum of six or eight characters is barely adequate.

A strong mixture of characters for a password includes at least one capital letter, at least one lower case letter, at least on digit, and at least one punctuation character. You should avoid repeating any character more than once in the same password. The special character (such as !@#$%^&*,;) should not be th efirst or last character in the password.

Avoid using any word that occurs in your own or any other natural langauge. Hackers use a **dictionary attack** that tries words from the dictionary. Also avoid spelling words backwards, using common misspellings, or using abbreviations. Avoid using dates that are important to you (someone can easily look up your birthday or anniversary on the world wide web). Avoid using names of family, friends, or even pets.

secure technique

A technique that generates decent passwords is to use a key phrase and then use the first letter of each word in the keyword. Sprinkle in digits and special characters (punctuation) and make some of the letters upper case and some lower case.

Never use the same password for more than one purpose. People have the tendency to reuse the same password over and over. If a hacker gets your password from one system, the hacker will see if it also works on your bank account and other systems.

superuser

The super user (root) can use the `passwd` command to reset any other user's password. There is no prompt for the current (old) password.

```
$ passwd username
```

The super user (root) can also remove a password for a specific user with the `-d` option. The disable option then allows the specified user to login without a password.

```
$ passwd -d username
```

command structure

chapter 9

summary

This chapter looks at simple UNIX/Linux commands to get you started with using the shell.

You will learn the basic format or structure of a shell command.

You can run a UNIX (or Linux or Mac OS X) command (also called a tool) by typing its name and then the ENTER or RETURN key.

single command

The most simple form of a UNIX command is just a single command by itself. Many UNIX/Linux commands will do useful work with just the command by itself. The next examples show two commands (`who` and `date`) by themselves.

The output generated by a single command by itself is called the default behavior of that command.

who

The `who` command will tell you all of the users who are currently logged into a computer. This is not particularly informative on a personal computer where you are the only person using the computer, but it can be useful on a server or a large computing system.

Type `who` followed by the ENTER or RETURN key.

```
$ who
admin   console Aug 24 18:47
admin   ttys000 Aug 24 20:09
$
```

The format is the login name of the user, followed by the user’s terminal port, followed by the month, day, and time of login.

failed command

The shell will let you know when you have typed something it doesn’t understand.

Type “asdf” and then the ENTER or RETURN key. You should see a message similar to the following:

```
$ asdf
-bash: asdf: command not found
$
```

date

The following example uses the **date** command or tool.

```
$ date
Wed Nov 10 18:08:33 PST 2010
$
```

The format is: day of the week, month, day of the month, 24 hour time, time zone, year.

options, switches, or flags

A command may optionally be followed by one or more options. The options are also commonly called flags or switches.

Options are usually a single character. Usually the option is preceded by a minus sign or hyphen character.

See the following example.

universal time

Adding the `-u` flag to the command `date` will cause it to report the time and date in UTC (Coordinated Universal) time (also known as Zulu Time and formerly known as Greenwich Mean Time or GMT). The seconds are slightly higher in the second example because of the passage of time.

```
$ date
Sat Aug 25 19:09:19 PDT 2012
$
$ date -u
Sun Aug 26 02:09:27 UTC 2012
$
```

arguments

Commands may also optionally have arguments. The most common arguments are the names of files.

Technically, the options just mentioned are also arguments, but in common practice options are separated from other arguments in discussions. Also, technically, the operators mentiond below are also arguments, but again it is useful to separate operators from other arguments in discussions.

The following shows the difference between the default behavior (no arguments) of `who` and a version with arguments `who am i`.

The default behavior of `who` lists all users on the computer.

```
$ who
admin   console Aug 24 18:47
admin   ttys000 Aug 24 20:09
$
```

The use of the arguments with `who am i` causes the command to only lists the one user who typed the command.

The command is `who` and the arguments are `am i`.

```
$ who am i
admin    ttys000  Aug 25 17:30
$
```

one argument

In an upcoming chapter on the `cat` you will use the command `cat` with the file name `file01.txt` to confirm that you correctly entered your sample file.

just observe this example
do *not* type this into your shell

```
$ cat file01.txt
This is a line of text in my first file.
This is another line.

To be, or not to be: that is the question:

1234567890
ABC
XYZ
abc
xyz
$
```

You may download this sample file from http://www.osdata.com/programming/shell/file01.txt.

two arguments

In the upcoming chapter on the `cat` you will use the command `cat` with two file names (`file01.txt` and `numberfile.txt`) to confirm that you correctly entered two of your sample files.

just observe this example
do *not* type this into your shell

```
$ cat file01.txt numberfile.txt
This is a line of text in my first file.
This is another line.

To be, or not to be: that is the question:

1234567890
ABC
XYZ
abc
xyz
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
$
```

You may download the second sample file from http://www.osdata.com/programming/shell/numberfile.txt.

options and arguments

You can combine options and arguments on the same command line.

The following example uses the command `cat` with the `-b` option and the file name `file01.txt`.

just observe this example
do *not* type this into your shell

```
$ cat -b file01.txt
     1  This is a line of text in my first file.
     2  This is another line.

     3  To be, or not to be: that is the question:

     4  1234567890
     5  ABC
     6  XYZ
     7  abc
     8  xyz
$
```

operators and special characters

Command lines may include optional operators or other special characters.

In an upcoming chapter on the `cat` you will use the command `cat` with the operator > and the file name `file01.txt` to enter your sample file.

just observe this example
do *not* type this into your shell

```
$ cat > file01
```

If you accidentally typed in the above command, hold down the CONTROL key and the D key at the same time to return to your shell.

quick tour of shell commands

chapter 10

summary

This chapter gives you a quick tour of shell commands.

You should do each of the exercises and observe what happens.

Do not worry about trying to learn how each of the commands works.

The goal here is to give you a basic idea of how a shell works and the kinds of things that the most common shell commands and operations do.

This quick tour will give you a background to understand the upcoming chapters and their lessons. If you have the need to use anything you observe in this quick tour, you can always jump ahead to the appropriate chapter and learn the details.

echo text

Use the `echo` command to send text to the terminal. Remember to press the ENTER or RETURN key after each command. Note that this will be particularly useful in your future UNIX/Linux scripts to report information back from your scripts.

```
$ echo hello world
hello world
$
```

echo text with variable information

Use the `echo` command with an environment variable. it should use your account name.

```
$ echo hello $USER
hello admin
$
```

list directory contents

Use the `ls` command to list the contents of the current directory.

```
$ ls
Desktop         Movies        Send registration
Documents       Music         Sites
Downloads       Pictures
Library         Public
$
```

create a text file

Use the `cat` command to create a small text file.

Type the command line, followed by RETURN or ENTER, then type each of the six suggested lines, each followed by the RETURN KEY. After having enetered each line, make sure you are at the beginning of a new (blank or empty) line and type Ctrl-D (hold downt he CONTROL key and the D key at the same time).

```
$ cat > names
James
Mary
John
Patricia
```

```
Robert
Linda
CONTROL-D
$
```

The choice of names is based on the most popular names in the United States in the year before this chapter was written. Bribe me if you want your name put into the example.

check the file was created

Use `ls` to make sure the file was created properly. It should be added to your directory listing.

```
$ ls
Desktop          Movies        Send registration
Documents        Music         Sites
Downloads        Pictures      names
Library          Public
$
```

display file contents

Use the `cat` command to show the contents of your new file.

```
$ cat names
James
Mary
John
Patricia
Robert
Linda
$
```

count the number of words

Use the `wc` command to count the number of words in your new file.

```
$ wc names
       6       6      38 names
$
```

The format is the number of lines (`6`), followed by the number of words (`6`), followed by the number of characters (`38`), followed by the name of the file (`names`).

copy a file

Use the `cp` command to make a copy of a file.

```
$ cp names saved_names
$
```

Notice that there is no confirmation of the file copy being made.

This silent behavior is typical of any UNIX shell. The shell will typically report errors, but remain silent on success. While disconcerting to those new to UNIX or Linux, you become accustomed to it. The original purpose was to save paper. When UNIX was first created, the terminals were mostly teletype machines and all output was printed to a roll of paper. It made sense to conserve on paper use to keep costs down.

You can use the `ls` command to confirm that the copy really was made. You won't be using up any paper.

```
$ ls
Desktop          Movies        Send registration
Documents        Music         Sites
```

```
Downloads        Pictures      names
Library          Public        saved_names
$
```

rename a file

Use the mv command to rename a file.

```
$ mv saved_named old_names
$
```

Notice that the shell is once again silent with success. You can use the ls command to confirm that the rename really was made.

```
$ ls
Desktop          Movies        Send registration
Documents        Music         Sites
Downloads        Pictures      names
Library          Public        old_names
$
```

delete a file

Use the rm (remove) command to delete a file.

```
$ rm old_names
$
```

You can use the ls command to confirm that the file was really deleted.

```
$ ls
Desktop          Movies        Send registration
Documents        Music         Sites
Downloads        Pictures      names
Library
$
```

current directory

Use the pwd command to determine the current directory. Your version should give the name of your home directory, which normally matches the name of your user account. The example include the directory as part of the prompt (your system may not include this) and the tilde (~) character indicates the home directory.

```
$ pwd
/Users/admin
admins-power-mac-g5:~ admin$
```

make a directory

Use the mkdir command to make a new directory (folder).

```
$ mkdir testdir
admins-power-mac-g5:~ admin$
```

change directory

Use the cd command to change to your new directory (folder). if your prompt includes the current directory, then you will see your prompt change to show the new location.

```
$ cd testdir
admins-power-mac-g5:testdir admin$
```

confirm directory change

Use the pwd command to confirm that you are now in your new directory.

```
$ pwd
/Users/admin/testdir
admins-power-mac-g5:testdir admin$
```

return to home directory

Use the cd command without any additional arguments to return to your home directory from anywhere.

```
$ cd
admins-power-mac-g5:~ admin$
```

confirm directory change

Use the pwd command to confirm that you are now back in your home directory.

```
$ pwd
/Users/admin
admins-power-mac-g5:~ admin$
```

Now you are ready to dive in and start becoming proficient at using the UNIX or Linux shell.

man

chapter 11

summary

This chapter looks at `man`, a UNIX (and Linux) command.

`man` is the UNIX equivalent of a help function.

`man` is the command used to view manual pages.

This chapter show you how to use the `man` command to get help. Because of huge variation in the various flavors of UNIX and Linux, the `man` installed on your computer or server is the best source of detailed information. If you have trouble with any of the lessons in this book or in general use of your UNIX or Linux system, always refer to the local `man` for help.

UNIX was the first operating system distributed with online documentation. This included `man` (manual pages for each command, library component, system call, header file, etc.) and doc (longer documents detailing major subsystems, such as the C programming language and `troff`.

example of man command with options

Most BASH commands accept options. These options follow a space character and are typed before you press RETURN or ENTER.

The format is `man` is followed by the name of the command or tool that you want to view. You will get the manual pages for the named command or tool.

The following example uses the **man** command or tool.

```
      $ man date
DATE(1)                  BSD General Commands Manual                 DATE(1)

NAME
      date -- display or set date and time

SYNOPSIS
      date [-ju] [-r seconds] [-v [+|-]val[ymwdHMS]] ... [+output_fmt]
      date [-jnu] [[[mm]dd]HH]MM[[cc]yy][.ss]
      date [-jnu] -f input_fmt new_date [+output_fmt]
      date [-d dst] [-t minutes_west]
DESCRIPTION
      When invoked without arguments, the date utility displays the current
      date and time. Otherwise, depending on the options specified, date will
      set the date and time or print it in a user-defined way.

      The date utility displays the date and time read from the kernel clock.
      When used to set the date and time, both the kernel clock and the hard-
      ware clock are updated.

      Only the superuser may set the date, and if the system securelevel (see
      securelevel(8)) is greater than 1, the time may not be changed by more
      than 1 second.
:
```

Typing the RETURN or ENTER key will bring up the next line of the manual page.

Typing the SPACE-BAR key will bring up the next page of text.

Typing the UP-ARROW or DOWN-ARROW key will move up or down one line.

Typing **q** will end viewing the manual page and return to the BASH prompt (the entire manual page will disappear from view).

The **man** command or tool will be a very useful reference.

A `man` page is typically organized:

`NAME:` Command name and brief description.

`SYNOPSIS:` The syntax for using the command, along with the flags (options) the command takes.

`DESCRIPTION:` Details about the command.

`ENVIRONMENT:` The environment variables used by the command.

`EXIT STATUS:` Information about how the command reports errors or success.

`FILES:` File related to the command.

`SEE ALSO:` related commands.

`STANDARDS:` The international standards, if any.

Some systems might have an `AUTHOR` or other sections.

Type `man man` for information about the local version of `man`.

man sections

The first seven distributions of UNIX (called V1 UNIX through V7 UNIX and collectively called traditional UNIX) included a two volume printed manual, which was divided into eight sections.

Some manual pages are located in sections. There are generally eight sections:

1. General commands
2. System [kernel] calls
3. C library functions
4. Special files (such as devices) and drivers
5. File formats, conventions, and miscellaneous information
6. games and screensavers
7. Macro packages
8. System administration commands and daemons

Note that on some systems, the system administration commands are in section 1m. This section is also sometimes called the Maintenance commands.

Note that on some systems, section 7 is Miscellaneous.

You can look at a particular section by adding the section number to the command line.

```
$ man SECTION-NUMBER commandname
```

You can use the `whatis` command to find the sections and then look at the particular section of your choice.

```
$ whatis crontab
$ whatis crontab
crontab(1)              - maintain crontab files for individual users (V3)
crontab(5)              - tables for driving cron

$ man 5 crontab
```

cat

chapter 12

summary

This chapter looks at `cat`, a UNIX (and Linux) command.

The `cat` (as in con**cat**enate) utility can be used to concatenate several files into a single file.

`cat` is most often used to obtain the contents of a file for input into a Linux or UNIX shell script.

`cat` is used concatenate file. The name is an abbreviation of catenate, a synonym of concatenate.

create names file

If you did not create the names file suggested in the quick tour chapter, please do so now, because you will use this file in future exercises. If you already created the names file, then you can safely skip this step.

Type the command line, followed by RETURN or ENTER, then type each of the six suggested lines, each followed by the RETURN KEY. After having enetered each line, make sure you are at the beginning of a new (blank or empty) line and type Ctrl-D (hold downt he CONTROL key and the D key at the same time).

```
$ cat > names
James
Mary
John
Patricia
Robert
Linda
CONTROL-D
$
```

The choice of names is based on the most popular names in the United States in the year before this subchapter was written. Bribe me if you want your name put into the example.

check the file was created

Use `ls` to make sure the file was created properly. It should be added to your directory listing.

```
$ ls
Desktop          Movies        Send registration
Documents        Music         Sites
Downloads        Pictures      names
Library          Public
$
```

display file contents

Use the `cat` command to show the contents of your new file.

```
$ cat names
James
Mary
John
Patricia
Robert
Linda
$
```

creating files

One simple use of `cat` is to create simple files.

Type `cat > file01.txt`, followed by ENTER or RETURN.

```
$ cat > file01.txt
```

The cursor will be at the beginning of a line that has no prompt. You are no longer in the shell, but instead in the `cat` utility. There is no `cat` prompt.

Type the following lines (by convention, all of the input should be in bold, but to make it easier on the eye, it is in italics here):

```
This is a line of text in my first file.
This is another line.

To be, or not to be: that is the question:

1234567890
ABC
XYZ
abc
xyz
```

Once you have typed in these lines, press RETURN to make sure you are at the beginning of a new line.

Press Ctrl-D (hold down the CONTROL key and the D key at the same time). This indicates end-of-file (EOF), which informs `cat` that you have no more input. This will return you to the shell.

Note that unlike some operating systems, the ".txt" file extension is *not* required by UNIX or Linux. It is optional, but the file extensions are often used to make it easier for a human to distinguish file types.

Now repeat the process with the following file (by convention, all of the input should be in bold, but to make it easier on the eye, it is in italics here):

There is a lot of typing in this example, but it is important for the upcoming exercises. Type the following lines (by convention, all of the input should be in bold, but to make it easier on the eye, it is in italics here):

Because of the length of this file, you may want to download a copy from the internet. There is a copy at http://www.osdata.com/programming/shell/file02.txt — if you know how to download a copy and place it in your home directory, save yourself some typing time.

```
$ cat > file02.txt

This is a line of text in my second file.
This is another line.

To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die: to sleep;
No more; and by a sleep to say we end
The heart-ache and the thousand natural shocks
That flesh is heir to, 'tis a consummation
Devoutly to be wish'd. To die, to sleep;
To sleep: perchance to dream: ay, there's the rub;
For in that sleep of death what dreams may come
When we have shuffled off this mortal coil,
Must give us pause: there's the respect
That makes calamity of so long life;
For who would bear the whips and scorns of time,
The oppressor's wrong, the proud man's contumely,
The pangs of despised love, the law's delay,
The insolence of office and the spurns
That patient merit of the unworthy takes,
When he himself might his quietus make
With a bare bodkin? who would fardels bear,
To grunt and sweat under a weary life,
But that the dread of something after death,
The undiscover'd country from whose bourn
No traveller returns, puzzles the will
And makes us rather bear those ills we have
Than fly to others that we know not of?
Thus conscience does make cowards of us all;
And thus the native hue of resolution
Is sicklied o'er with the pale cast of thought,
And enterprises of great pith and moment
With this regard their currents turn awry,
And lose the name of action. - Soft you now!
The fair Ophelia! Nymph, in thy orisons
Be all my sins remember'd.

1234567890
ABC
XYZ
abc
xyz
```

Now repeat the process with the following file (by convention, all of the input should be in bold, but to make it easier on the eye, it is in italics here — note the cheat method below):

```
$ cat > numberfile.txt
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Once again, use the RETURN key to enter a new line and then use Ctrl-D to exit `cat`.

If you want to cheat on making this file, you can use the following command on Mac OS X or BSD systems:

```
$ jot 20 1 > numberfile.txt
$
```

You will use these files in many of the exercises in this book.

PC-DOS equivalent

`cat` is the UNIX equivalent of the MS-DOS or PC-DOS command `TYPE`. You can add the PC-DOS equivalent to your shell session with the `alias` command. To make the change permanent, add the following line to the `.bashrc` file in your home directory. Note that if you add this PC-DOS/MS-DOS equivalent, only add the all upper case version, because the lower case `type` is an important UNIX command that you will also need.

```
$ alias TYPE="cat"
```

view a file with cat

Type `cat file 01`, followed by the RETURN or ENTER key. You should see the contents of your first file. The beginning of the file may scroll off the top of your screen (this is normal).

```
$ cat file01.txt
This is a line of text in my first file.
This is another line.

To be, or not to be: that is the question:

1234567890
ABC
XYZ
abc
xyz
$
```

combine files with cat

Type `cat file 02 numberfile.txt`, followed by the RETURN or ENTER key. This will show you the contents of both files, one immediately after the other. This is the use for which `cat` was named.

```
$ cat file02.txt numberfile.txt
This is a line of text in my second file.
This is another line.

To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die: to sleep;
No more; and by a sleep to say we end
The heart-ache and the thousand natural shocks
That flesh is heir to, 'tis a consummation
Devoutly to be wish'd. To die, to sleep;
    ...many text lines...
The fair Ophelia! Nymph, in thy orisons
Be all my sins remember'd.

1234567890
ABC
XYZ
abc
xyz
1
2
3
4
5
6
7
```

```
        8
        9
        10
        11
        12
        13
        14
        15
        16
        17
        18
        19
        20
        $
```

This should list your new file that combines combines two of the files you created. The beginning of the listing will scroll off the top of your screen. Don't worry about it. Well see how to view long files in the chapter about `less`.

display with line numbers

The following example uses the command `cat` with the `-b` option to add line numbers to the listing of the file.

Type `cat -b file01.txt`.

```
        $ cat -b file01.txt
            1  This is a line of text in my first file.
            2  This is another line.

            3  To be, or not to be: that is the question:

            4  1234567890
            5  ABC
            6  XYZ
            7  abc
            8  xyz
        $
```

command separator

chapter 13

This chapter looks at the command separator in UNIX or Linux.

The semicolon (**;**) is used as a command separator.

You can run more than one command on a single line by using the command separator, plaing the semicolon between each command.

```
$ date; who am i
Mon Aug 27 19:15:41 PDT 2012
admin    ttys000  Aug 27 17:50
$
```

It does not matter if you have a space before the semicolon or not. Use whichever method is more readable and natural for you.

```
$ date ; who am i
Mon Aug 27 19:15:41 PDT 2012
admin    ttys000  Aug 27 17:50
$
```

Each command is processed in order, as if you had typed each individually, with the exception that the line with the prompt is only displayed once, at the end of the series of commands.

If you forget the semicolon, you will get an error message. The following two examples are from different systems.

```
$ date who am i
date: illegal time format
$

$ date who am i
date: bad conversion
$
```

The semicolon is a common terminator or separator in many common programming languages, including Ada, C, Java, Pascal, Perl, PHP, and PL/I.

Many programmers automatically place the semicolon at the end of any shell command. The shell is fine with this.

```
$ date;
Mon Aug 27 19:15:41 PDT 2012
$
```

less, more, pg

chapter 14

summary

This subchapter looks at `less`, `more`, and `pg`, a related family of UNIX (and Linux) commands.

`less`, `more`, and `pg` are utilities for reading a very large text file in small sections at a time.

`pg` is the name of the historical utility on BSD UNIX systems.

`more` is the name of the historical utility on System V UNIX systems.

`pg` and `more` are very similar and have almost the same functionality and are used in almost the exact same manner, except for whether you type `pg` or `more`.

`less` is a more modern version that has the capaibilites of `more` along with additional new capabilities.

Mac OS X has all three versions installed. It is common to find both `more` and `less` on Linux systems. The older versions are typically included so that old scripts written before the invention of `less` will still work.

The name `less` is a pun, from the expression “less is more.”

Try using `less` with your test file `file01.txt`.

You are going to try `less` first.

```
$ less file01.txt
```

If that doesn’t work, you are going to try `more` next.

```
$ more file01.txt
```

If that doesn’t work, you are going to try `pg` last.

```
$ pg file01.txt
```

One of the three will work. You can try the others to see if they are available on your system, but for the exercise, use the first one that works.

```
$ less file01.txt
This is a line of text in my first file.
This is another line.

To be, or not to be: that is the question:

1234567890
ABC
XYZ
abc
xyz
$
```

`Now we are going to see the real power of less with a long file.`

```
$ less file02.txt
This is a line of text in my second file.
This is another line.

To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die: to sleep;
No more; and by a sleep to say we end
The heart-ache and the thousand natural shocks
That flesh is heir to, 'tis a consummation
Devoutly to be wish'd. To die, to sleep;
To sleep: perchance to dream: ay, there's the rub;
For in that sleep of death what dreams may come
When we have shuffled off this mortal coil,
Must give us pause: there's the respect
That makes calamity of so long life;
For who would bear the whips and scorns of time,
The oppressor's wrong, the proud man's contumely,
The pangs of despised love, the law's delay,
The insolence of office and the spurns
That patient merit of the unworthy takes,
When he himself might his quietus make
File02.txt
```

Notice that you are not yet back to the shell prompt. You are still in the less command. You are only one page into the file. less allows you to view long files one page at a time, instead of scrolling them off the top of your screen.

Type the SPACE key to see another page of text.

```
        That patient merit of the unworthy takes,
        When he himself might his quietus make
        With a bare bodkin? who would fardels bear,
        To grunt and sweat under a weary life,
        But that the dread of something after death,
        The undiscover'd country from whose bourn
        No traveller returns, puzzles the will
        And makes us rather bear those ills we have
        Than fly to others that we know not of?
        Thus conscience does make cowards of us all;
        And thus the native hue of resolution
        Is sicklied o'er with the pale cast of thought,
        And enterprises of great pith and moment
        With this regard their currents turn awry,
        And lose the name of action. - Soft you now!
        The fair Ophelia! Nymph, in thy orisons
        Be all my sins remember'd.

        1234567890
        ABC
        XYZ
        abc
        xyz
        (END)
```

Notice that you are not yet back to the shell prompt. Even though the entire file contents have been listed, you are still in the `less` command.

Typing lower case `q` will quit the `less` (or `more` or `pg`) program.

`less` (and the other two utility commands) displays one page at a time. This is in contrast with `cat`, which displays the entire file at once.

file system basics

chapter 15

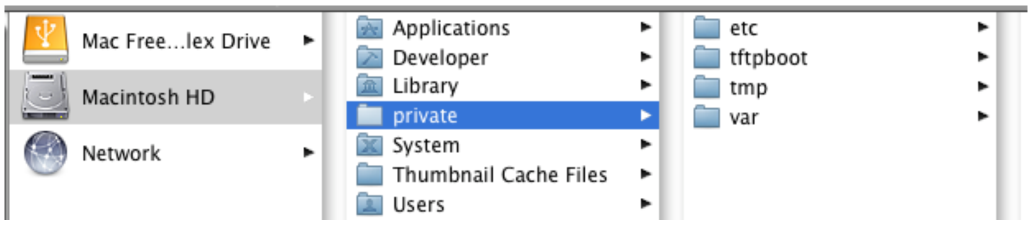
summary

This chapter looks at the basics of the file system on UNIX or Linux.

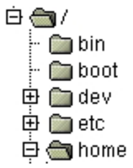
graphics example

In a graphic user interface, you can see the file system directory structure in some kind of picture version. This varies by system.

The following example is from Mac OS X. If you use Mac OS X, the UNIX directories are kept hidden in Finder. You can see them by using Finder's **Go** menu, Go To Folder…, and type "private".



The following example is from Linux.



Do **not** mess with your system files until you know what you are doing. You can cause your computer to stop working.

Some of the typical UNIX/Linux directories are etc, tmp, var, bin, sbin, dev, boot, or home.

some basics

Both UNIX and Linux organize the entire file system into a single collection (called the directory tree). This is in sharp contrast to Windows dividing the file system into multiple parts, each based on a particular drive or device.

File names are case sensitive. The file names "File" and "file" are different files. This is in sharp contrast with Windows and Macintosh (where they are alternate names for the same file). Note that this is a Mac OS X variation from the standard UNIX practice.

A file name that starts with a period is a hidden file. In Mac OS X, you can toggle between showing and hiding hidden files in Finder by typing Command-Shift-Period.

Neither UNIX nor Linux enforce or recognize file extensions (such as ".txt"). In some operating systems (such as Windows and traditional Macintosh), the file extensions are recognized by the operating system and are used for such purposes as deciding which program is used to open the file. You can add file extensions UNIX or Linux file names (and this is commonly done to help humans organize their files), but the file extension is ignored by the operating system and has no special meaning.

Most modern Linux and UNIX operating systems support long file names with a wide variety of characters, including the space character. Shells are set up for old school UNIX, which had a limited number of characters in file names and a very limited choice of characters in file names. it is best to use these old school limits for files you manipulate through the UNIX or Linux shell. Mac OS X allows you to put file names inside quotation marks (to allow such things as spaces in file names). You can also access files through their inode numbers (which will let the shell work on any file name). Still, it is wise to use only letters, numbers, period, hyphen, and underscore in UNIX or Linux file names 9and particularly to avoid the space character).

directory tree

A directory in UNIX or Linux is the same as a folder in Macintosh or Windows.

Directories may contain any combination of files and subordinate directories.

This structure is called a tree. It branches out from a root.

Unix and Linux are organized into a single file system tree, under a master directory called root and designated by the forward leaning slash (`/`). This is in sharp contrast with Windows, where there is a seperate file system tree for each drive letter.

In UNIX, everything is is a file. Even a directory is just a special file that stores information about other files.

some important directories

The following is a brief description of some of the major directories. Not all systems will have all of these directories.

/ The root directory of the entire file system.

/bin A collection of binary files, better known as programs.

/boot or **/kernel** The files needed to boot or start your computer.

/dev The devices connected to your computer (remember, everything, even hardware devices, is treated as a file in UNIX or Linux).

/etc System configuration files, startup procedures, and shutdown procedures.

/home or **/users** The home directories for each user.

/lib or **/Library** Library files.

/net Other networked systems (again, treated as files).

/opt Third party software. In older systems this might be **/usr/local**

/private On Mac OS X, this is the location of the UNIX files. It is normally kept hidden from all users in Finder, but is visible in your terminal emulator and shell. Mac OS X has links to the major UNIX directories at the root level so that UNIX and Linux tools can find their files in the standard locations.

/proc Information about processes and devices.

/sbin System binaries (that is, system programs).

/tmp Space for temporary files. required by all forms of UNIX.

/usr User binaries (programs), libraries, manuals, and docs.

/var Variable files.

Swap Virtual memory on a hard drive. Allows the memory manager to swap some data and program segments to hard drive to expand the amount of memory available beyond the limits of physical memory.

For a more advanced and complete listing, see major directories.

home and working directory

The home directory is the directory that your system places you in when you first log into your system (start the shell). The home directory is defined in the `/etc/passwd` file. This is your personal space on a UNIX or Linux machine.

The tilde character can be used to name your home directory. `~`*`user-name`* (where user-name is your actual user name) indicates your home directory.

The working directory is whatever directory you happen to be working in. The working directory is always the same as the home directory when you start a shell session, but you can change your working directory at any time (and typically do change it). Commands normally are applied to the current working directory.

The working directory may also be called the current directory, current working directory, or present working directory.

parent and child directory

Any directories underneath a particular directory are called child directories. Any directory can have zero, one, or many child directories.

The directory above any particular directory is called the parent directory. Any directory has only one parent (with the exception of root, which has no parent).

absolute paths

Every directory or file can be defined by a complete absolute path. The complete absolute path gives every level of directory, starting from the root. An absolute path name is sometimes called the full path name.

An example might be the `/usr/java/bin`. This would be a directory containing Java binaries. In this example, under the `/` root directory there is a directory called `usr` and inside that directory there is a directory called `java` and inside that directory there is a directory called `bin` and inside that directory there are programs for running Java.

Absolute paths normally start at the root of the file system.

relative paths

Relative paths describe where to find a file or directory from the current working directory.

A single period (or dot) character (`./`) indicates that you are describing the path to a directory or file from the current working directory.

If your current working directory is `www` and that directory is under `/usr`, then the file `index.html` inside the current working directory can be described by the full or absolute path of `/usr/www/index.html` or a relative path name of `./index.html`.

The single period is called “dot”.

Two periods (`../` called “dot dot”) take you up one directory to the parent directory.

You can use a series of two dots (`../../`) to go up more than one directory level.

dots and tildes and slashes

`/` Slash is used as a separator between directory names. When slash is the first character in a path name, it indicates the root directory.

~ Tilde is used to get to your home directory.

`./` Single dot is used to indicate the current working directory.

`../` Two dots are used to indicate the parent of the current working directory.

`.` A single dot in front of a file name (such as `.Trash`) is used to make a hidden file. Hidden files are only listed when specifically requested.

moving around

This will all make a bit more sense in the next chapter as you learn how to actually move around your file system.

pwd

chapter 16

summary

This chapter looks at `pwd`, a UNIX (and Linux) command.

`pwd` is used to Print the Working Directory. It provides the full path for the current working directory. and is normally run without any options.

`pwd` is a builtin command in `bash`. There is also an external utility with the same name and functionality.

working directory

The working directory is whatever directory you happen to be working in. The working directory is always the same as the home directory when you start a shell session, but you can change your working directory at any time (and typically do change it). Commands normally are applied to the current working directory.

The working directory may also be called the current directory, current working directory, or present working directory.

using pwd

Type pwd from the command line, followed by ENTER or RETURN, and you will see the current working directory.

```
$ pwd
/Users/admin
$
```

There's not much else to tell you about this command, but this is an important one because you will continually have the need to know where you are.

moving around

We will go into more detail about these commands later, but for now you need to know how to move around in your file system.

The `ls` command is used to list the contents of the current directory.

Go ahead and type `ls` followed by ENTER or RETURN and you will see a list of the files and directories in your current working directory (which should still also be your home directory).

The `cd` command is used to change directories.

Go ahead and type `cd` *`directory-name`* followed by ENTER or RETURN and you will move to the new directory. You can confirm this by then typing `pwd` followed by ENTER or RETURN. You will see that you have successfully changed working directories.

```
$ pwd
/Users/admin
$ ls
Desktop     Movies      Send Registration
Documents   Music       Sites
Downloads   Pictures
Library     Public
$ cd Desktop
$ pwd
/Users/admin/Desktop
$
```

As you move around the directory tree, it is common to forget exactly where you are, especially if you do some other tasks (such as file editing). `pwd` provides an easy way to determine where you are and insure that your next actions are happening in the correct directory.

Use `cd` command all by itself to return to your home directory.

```
$ cd Desktop
$ pwd
/Users/admin
$
```

options

`pwd` is almost always used without options and never uses any arguments (such as file names). Anything other tahn legal options typed after `pwd` are ignored. Some shells ignore anything other than the legal options typed after `pwd` without reporting any errors, while most shells report errors for illegal options. All shells seem to ignore all arguments without reporting any errors.

The two main options for `pwd` are `-L` and `-P`.

The `-L` option displays the logical current working directory. This is the default if no options are listed.

```
$ pwd -L
/Users/admin
$
```

The `-P` option displays the physical current working directory, with all symbolic links resolved.

```
$ pwd -P
/Users/admin
$
```

You can use the `help` command to get information about `pwd`.

```
$ help pwd
```

Some early Linux shells allow the *--help* command to get information about `pwd`.

```
$ pwd --help
```

Some early Linux shells allow the *--version* option gives the version number for your copy of `pwd`.

```
$ pwd --version
```

command history

chapter 17

summary

This chapter looks at command history in a UNIX (and Linux) BASH shell.

Command history is used to recall recent commands and repeat them with modifications.

You will find that you are often repeating a few common commands over and over with variations in the options and arguments (such as file names). Having the ability to recall a previous example and make a few changes will greatly save on typing.

arrow keys

Pushing the Up Arrow key will bring back previously typed commands one at a time (from most recently typed to the first typed).

Pushing the Down Arrow key will move from the oldest to the newest commands one at a time.

The commands brought back by the Up Arrow and Down Arrow keys will appear after the current prompt.

You can press the ENTER or RETURN key to immediately run any selected command.

After using the Up Arrow or Down Arrow keys, your cursor is at the end of the line. You can use the DELETE key to remove old arguments and then start typing in new ones.

Note that holding down the CONTROL key and pressing the p key at the same time will take you to the previous command just like the Up Arrow key. This is written ^p.

Holding down the CONTROL key and pressing the n key at the same time will take you to the next command just like the Down Arrow key. This is written ^n.

Pressing the META key (or on some systems pressing the ESCAPE key), then releasing it, then typing the less than < key will take you to the beginning of the command history. Pressing the META key (or on many modern systems pressing the ESCAPE key or ESC key), then releasing it, then typing the greater than > key will also move to the end (most recent) command history. These are written M-< and M->.

editing a line

As mentioned above, when you use command history, the cursor is at the end of the recalled command.

Either the Left Arrow or ^b will go one character backward (towards the beginning of line).

Either the Right Arrow or ^f will go one character forward (towards the end of line).

Typing inside a line will place characters in front of the cursor. The character inside the cursor remains unchanged.

Using the DELETE key will delete the character in front of the cursor. The character inside the cursor remains unchanged.

^k will delete everything from the cursor to the end of the line.

^d will delete the character inside (under) the cursor.

M-b will move you one word back (towards beginning of line). In this case, a word is a collection of characters, usually separated by a space character.

M-f will move you one word forward (towards end of line).

^e will move to the end of the line and ^a will move to the beginning of the line.

Once you have edited your command, use the RETURN or ENTER key to run the new command. You can use the ENTER or RETURN key from anywhere in the command line. You do not have to be at the end of the command line.

built-in commands

chapter 18

summary

This chapter looks at built-in commands in a UNIX (and Linux) BASH shell.

tools

Most of the commands in the UNIX or Linux shell are actually programs. If you look at the `usr/bin` directory, you will see the actual programs. Most of these programs were written in the C programming language.

There is a common core of tools/commands that will be available on almost every UNIX or Linux machine, but exactly how many and which commands/tools are available varies widely.

The good news is that if a command or tool is missing from your system, you can go out and get the source code and recompile it for your local computer.

built-in

Many of the shells have special built-in commands. These are *not* separate programs, but are part of the code for the shell itself.

One example would be the shell command `cd` that you just saw in the previous quick tour chapter.

There are some built-in commands are only available in selected shells and these can make your scripts shell-dependent.

Some examples of built-in commands include the `history` command in the C shell, and the `export` command in the Bourne shell. The `cd` command is built-in in both `bash` and `csh`.

`echo` is an example of a command that is built into both `bash` and `csh`, but also exists externally as a utility.

overriding built-in commands

You can override any built-in commands by giving the full path name to an external command or utility. If `bash` finds a slash character ( ) anywhere in a command, the shell will not run the built-in command, even if the last component of the specified command matches the name of a builtin command.

As an example, using the command `echo` will run the version of the command that is built into `bash`, while specifying `/bin/echo` or `./echo` will ignore the built-in comamnd and instead run the designated utility.

Overriding can be used to run alternative versions of commands or to extend the built-in command to add additional features.

determining builtin or external

You can use the `type` command to determine if a particular command is a built-in command or an external utility. If the command is an external utility, you will also be told the path to the external command.

```
$ type echo
echo is a shell builtin
$ type mkdir
mkdir is /bin/mkdir
$
```

You can use the `which` command to locate a program in your path.

```
$ which echo
/bin/echo
$
```

You can use the `whereis` command to locate a program in your path.

```
$ whereis echo
/bin/echo
$
```

In `csh` and `tcsh` you can use the `where` command to locate a program in your path.

```
% where echo
/bin/echo
%
```

problems

You can use the `type` command to determine if a particular command is a built-in command or an external utility. If the command is an external utility, you will also be told the path to the external command.

If something bad happens to your computer, if the shell is still loaded in memory and running, any of the built-in commands will still work correctly, even if the entire file system (including all hard drives) disappears or becomes unavailable for any reason.

built in command chart

The following chart shows the built-in commands and external utilities for `bash` and `csh` for Mac OS X. This wil be similar for Linux and other UNIXes.

External commands marked No** under the External column do exist externally, but are implemented using the built-in command.

Command	External	csh	bash
`!`	No	No	Yes
`%`	No	Yes	No
`.`	No	No	Yes
`:`	No	Yes	Yes
`{`	No	No	Yes
`}`	No	No	Yes
`alias`	No**	Yes	Yes
`alloc`	No	Yes	No
`bg`	No**	Yes	Yes
`bind`	No	No	Yes
`bindkey`	No	Yes	No
`break`	No	Yes	Yes
`breaksw`	No	Yes	No
`builtin`	No	No	Yes
`builtins`	No	Yes	No
`case`	No	Yes	Yes
`cd`	No**	Yes	Yes
`chdir`	No	Yes	Yes
`command`	No**	No	Yes
`complete`	No	Yes	No
`continue`	No	Yes	Yes
`default`	No	Yes	No
`dirs`	No	Yes	No
`do`	No	No	Yes
`done`	No	No	Yes
`echo`	Yes	Yes	Yes
`echotc`	No	Yes	No

`elif`	No	No	Yes
`else`	No	Yes	Yes
`end`	No	Yes	No
`endif`	No	Yes	No
`endsw`	No	Yes	No
`esac`	No	No	Yes
`eval`	No	Yes	Yes
`exec`	No	Yes	Yes
`exit`	No	Yes	Yes
`export`	No	No	Yes
`false`	Yes	No	Yes
`fc`	No**	No	Yes
`fg`	No**	Yes	Yes
`filetest`	No	Yes	No
`fi`	No	No	Yes
`for`	No	No	Yes
`foreach`	No	Yes	No
`getopts`	No**	No	Yes
`glob`	No	Yes	No
`goto`	No	Yes	No
`hash`	No	No	Yes
`hashstat`	No	Yes	No
`history`	No	Yes	No
`hup`	No	Yes	No
`if`	No	Yes	Yes
`jobid`	No	No	Yes
`jobs`	No**	Yes	Yes
`kill`	Yes	Yes	No
`limit`	No	Yes	No
`local`	No	No	Yes
`log`	No	Yes	No
`login`	Yes	Yes	No
`logout`	No	Yes	No
`ls-F`	No	Yes	No
`nice`	Yes	Yes	No
`nohup`	Yes	Yes	No
`notify`	No	Yes	No
`onintr`	No	Yes	No
`popd`	No	Yes	No
`printenv`	Yes	Yes	No
`pushd`	No	Yes	No
`pwd`	Yes	No	Yes
`read`	No**	No	Yes
`readonly`	No	No	Yes
`rehash`	No	Yes	No
`repeat`	No	Yes	No
`return`	No	No	Yes
`sched`	No	Yes	No
`set`	No	Yes	Yes
`setenv`	No	Yes	No

`settc`	No	Yes	No
`setty`	No	Yes	No
`setvar`	No	No	Yes
`shift`	No	Yes	Yes
`source`	No	Yes	No
`stop`	No	Yes	No
`suspend`	No	Yes	No
`switch`	No	Yes	No
`telltc`	No	Yes	No
`test`	Yes	No	Yes
`then`	No	No	Yes
`time`	Yes	Yes	No
`times`	No	No	Yes
`trap`	No	No	Yes
`true`	Yes	No	Yes
`type`	No	No	Yes
`ulimit`	No	No	Yes
`umask`	No**	Yes	Yes
`unalias`	No**	Yes	Yes
`uncomplete`	No	Yes	No
`unhash`	No	Yes	No
`unlimit`	No	Yes	No
`unset`	No	Yes	Yes
`unsetenv`	No	Yes	No
`until`	No	No	Yes
`wait`	No**	Yes	Yes
`where`	No	Yes	No
`which`	Yes	Yes	No
`while`	No	Yes	Yes

ls

chapter 19

summary

This chapter looks at `ls`, a UNIX (and Linux) command.

`ls` is the UNIX (and Linux) list command.

list directory contents

Use the `ls` command to list the contents of the current working directory.

```
$ ls
Desktop          Movies        Send registration
Documents        Music         Sites
Downloads        Pictures
Library          Public
$
```

PC-DOS equivalent

`ls -la` is the UNIX equivalent of the MS-DOS or PC-DOS command `DIR`. You can add the PC-DOS equivalent to your shell session with the `alias` command. To make the change permanent, add the following line to the `.bashrc` file in your home directory.

```
$ alias dir="ls -la"
$ alias DIR="ls -la"
```

`ls -la | less` is the UNIX equivalent of the MS-DOS or PC-DOS command `DIR /P`.

list all

Type `ls` with the `-A` option to get a list of all entries other than `.` (single dot) and `..` (double dot).

```
$ ls -A
```

list with hints

Type `ls` with the `-F` option to have special indicators for special files.

```
$ ls -F
Desktop/               Music/                 file01.txt
Documents/             Pictures/              file02.txt
Downloads/             Public/                names
Library/               Send registration@     numberfile.txt
Movies/                Sites/                 testdir/
$
```

A slash / will be added after every pathname that is a directory. An asterisk * will be added after every file that is executable (including scripts). An at sign @ will be displayed after each symbolic link. An equals sign = will be displayed after each socket. A percent sign % will be displayed after each whiteout. A vertical bar | will be displayed after each item than is a FIFO.

`ls-F` is a builtin command in `csh`.

cd

chapter 20

summary

This chapter looks at `cd`, a UNIX (and Linux) command.

`cd` is used to Change Directory.

`cd` is a builtin command in `bash` and `csh`. There is also an external utility with the same name and functionality.

`cd` is used to change the directory you are working in. You type the command `cd` followed by a space and then the directory (folder) that you want to change to.

change directory example

For the purposes of an example, we need to first find a directory to change to. Type the command `ls -F`.

```
$ ls -F
Desktop/                Music/                  file01.txt
Documents/              Pictures/               file02.txt
Downloads/              Public/                 names
Library/                Send registration@      numberfile.txt
Movies/                 Sites/                  testdir/
$
```

Your listing will be different than this one. Look for any name that has a slash (`/`) after it and use it as the *directory_name* in the following example. If you created the testdir in the quick tour chapter, then use the second example listed here.

```
$ cd directory_name
admins-power-mac-g5:directory_name admin$
```

If you created the testdir in the quick tour chapter, then use the following example:

```
$ cd testdir
admins-power-mac-g5:testdir admin$
```

Use the `pwd` command to confirm that you are now in your new directory.

```
$ pwd
/Users/admin/testdir
admins-power-mac-g5:testdir admin$
```

Use the `cd` command without any additional arguments to return to your home directory from anywhere.

```
$ cd
admins-power-mac-g5:~ admin$
```

Use the `pwd` command to confirm that you are now back in your home directory.

```
$ pwd
/Users/admin
admins-power-mac-g5:~ admin$
```

return to home directory

You can always return to your home directory from anywhere by typing `cd` command all by itself.

```
$ cd
admins-power-mac-g5:~ admin$
```

PC-DOS equivalent

`cd ..` is the UNIX equivalent of the MS-DOS or PC-DOS command `CD..`. You can add the PC-DOS equivalent to your shell session with the `alias` command. To make the change permanent, add the following line to the `.bashrc` file in your home directory.

```
$ alias cd..="cd .."
$ alias CD..="cd .."
```

cp

chapter 21

summary

This chapter looks at `cp`, a UNIX (and Linux) command.

`cp` is used to copy a file.

copy a file

Use the `cp` command to make a copy of a file. This example assumes you created the names file in the quick tour chapter.

```
$ cp names saved_names
$
```

Notice that there is no confirmation of the file copy being made.

This silent behavior is typical of any UNIX shell. The shell will typically report errors, but remain silent on success. While disconcerting to those new to UNIX or Linux, you become accustomed to it. The original purpose was to save paper. When UNIX was first created, the terminals were mostly teletype machines and all output was printed to a roll of paper. It made sense to conserve on paper use to keep costs down.

You can use the `ls` command to confirm that the copy really was made. You won't be using up any paper.

```
$ ls
Desktop          Movies        Send registration
Documents        Music         Sites
Downloads        Pictures      names
Library          Public        saved_names
$
```

`cp` makes an exact copy of a file.

PC-DOS equivalent

`cp -i` is the UNIX equivalent of the MS-DOS or PC-DOS command `COPY`. You can add the PC-DOS equivalent to your shell session with the `alias` command. To make the change permanent, add the following line to the `.bashrc` file in your home directory. Note that this version adds interactive questioning before replacing a file, in a manner similar to PC-DOS.

```
$ alias copy="cp -i"
$ alias COPY="cp -i"
```

mv

chapter 22

summary

This chapter looks at `mv`, a UNIX (and Linux) command.

`mv` is used to move or rename a file.

rename a file

Use the `mv` command to rename a file. This example assumes you created the names and saved-name files in the quick tour chapter.

```
$ mv saved_named old_names
$
```

Notice that the shell is once again silent with success. You can use the `ls` command to confirm that the rename really was made.

```
$ ls
Desktop          Movies        Send registration
Documents        Music         Sites
Downloads        Pictures      names
Library          Public        old_names
$
```

PC-DOS equivalent

`mv` is the UNIX equivalent of the MS-DOS or PC-DOS command `REN`. You can add the PC-DOS equivalent to your shell session with the `alias` command. To make the change permanent, add the following line to the `.bashrc` file in your home directory.

```
$ alias ren="mv"
$ alias REN="mv"
```

rm

chapter 23

summary

This chapter looks at `rm`, a UNIX (and Linux) command.

`rm` removes or deletes files or directories.

`rm` is used to remove or erase a file.

delete a file

Use the `rm` (remove) command to delete a file. This example assumes you created the names saved_names and old_names files in the quick tour chapter.

```
$ rm old_names
$
```

You can use the `ls` command to confirm that the file was really deleted.

```
$ ls
Desktop          Movies        Send registration
Documents        Music         Sites
Downloads        Pictures      names
Library
$
```

PC-DOS equivalent

`rm -i` is the UNIX equivalent of the MS-DOS or PC-DOS command `DEL`. You can add the PC-DOS equivalent to your shell session with the `alias` command. To make the change permanent, add the following line to the `.bashrc` file in your home directory. Note that this version adds interactive questioning before deleting a file, in a manner similar to PC-DOS.

```
$ alias del="rm -i"
$ alias DEL="rm -i"
```

recursive

Type `rm` followed by the option `-r` followed by *directory name* to delete recursively. This means that the directory and all of its subdirectories will be removed, even for non-empty direcories. This is a fast way to remove directories and files in a single command.

```
$ rm -r directory-name
$
```

sysadmin and root/superuser

chapter 24

summary

This chapter looks at system administration and the root or superuser account.

As we look at more advanced features and commands, we need to become aware of the root or superuser account and capabilities and how to safely access that power.

sysadmin

In the UNIX world, the system administrator is often called the sysadmin.

root/superuser

The root or superuser account has total authority to do anything on the system. This power is great for fixing problems, but bad because one accidenttally mistyped character could be very destructive. Some systems also have admin accounts of similar power.

Because of the potential for destructiveness, system administrators typically login with either a normal user account or a limited admin account for every day work and then switch to superuser or root only when absolutely necessary (and then immediately switch back).

The UNIX command for temporarily switching to root or superuser power is the `sudo` command, discussed in the next chapter.

special powers

The root or superuser account has powers that “mere mortal” accounts don’t have.

The root account has access to commands that effect the entire computer or system, such as the ability to halt or shutdown the system.

The root account is not affected by read and write file access permissions. The root or superuser account can create, remove, read, or write any file anywhere on the system.

Some commands have built-in restrictions that the root or superuser can ignore. For example, the system administrator can change any user’s password without knowing the old password.

sudo

chapter 25

summary

This chapter looks at `sudo`, a UNIX (and Linux) command.

sudo

The `sudo` command allows you to run a single command as another user, including at superuser or root level from a normal account. You will be asked for the password before the command will actually run.

This keeps you firmly in a normal account (with less danger of catastrophic failures), while still giving easy access to root or superuser power when really needed.

The `sudo` program was originally written by Bob Coggeshall and Cliff Spencer in 1980 at the Department of Computer Science at SUNY/Buffalo.

To run a single command as superuser or root, type `sudo` followed by a command.

```
$ sudo command
```

On Mac OS X the `sudo` command will fail if your account has no password.

On Mac OS X the `sudo` commands `password` prompt will not display anything (not even bullets or asterisks) while you type your password.

You will not be asked for a password if you use `sudo` from the root or superuser account. You will not be asked for a password if you use `sudo` and the target user is the same as the invoking user.

Some systems have a timer set (usually five minutes). You can run additional `sudo` commands without a password during the time period.

run in root shell

To change to in the root shell, type `sudo` followed by the option `-s`. The following warning is from Mac OS X (entered a root shell and then immediately returned to the normal shell). Note the change to the pound sign (#) prompt.

```
$ sudo -s

WARNING: Improper use of the sudo command could lead to data loss
typing when using sudo. Type "man sudo" for more information.

To proceed, enter your password, or type Ctrl-C to abort.

Password:
bash-3.2# exit
$
```

other users

To run a command as another user, type `sudo` followed by the option `-u` followed by the user account name followed by a command.

```
$ sudo -u username command
```

To view the home directory of a particular user:

```
$ sudo -u username ls ~username
```

edit files as www

To edit a file (this example is for index.html) as user www:

```
$ sudo -u www vim ~www/htdocs/index.html
```

which password

On some systems, you will authenticate with your own password rather than with the root or superuser password. The list of users authorized to run `sudo` are in the file /etc/sudoers (on Mac OS X, /private/etc/sudoers).

unreadable directories

To view unreadable directories:

```
$ sudo ls /usr/local/protected
```

shutdown

To shutdown a server:

```
$ sudo -r +15 "quick reboot"
```

usage listing

To make a usage listing of the directories in the /home partition (note that this runs the commands in a sub-shell to make the `cd` and file redirection work):

```
$ sudo sh -c "cd /home ; du -s * | sort -rn > USAGE"
```

security

The system can be set up to send a mail to the root informing of unauthorized attempts at using `sudo`.

The system can be set up to log both successful and unsuccessful attempts to `sudo`.

Some programs (such as editors) allow a user to run commands via shell escapes, avoiding `sudo` checks. You can use `sudo`'s `noexec` functionality to prevent shell escapes.

su

chapter 26

summary

This chapter looks at `su`, a UNIX (and Linux) command.

`su` is used to switch user accounts.

switch to another user account

You can use the `su` command to switch another user account. This allows a system administrator to fix things signed in to a particular user's account rather than from the root account, including using the user's settings and privileges and accesses rather than those of root.

who

chapter 27

summary

This chapter looks at `who`, a UNIX (and Linux) command.

list of who is using a system

The `who` command will tell you all of the users who are currently logged into a computer. This is not particularly informative on a personal computer where you are the only person using the computer, but it can be useful on a server or a large computing system.

Type `who` followed by the ENTER or RETURN key.

```
$ who
admin   console Aug 24 18:47
admin   ttys000 Aug 24 20:09
$
```

The format is the login name of the user, followed by the user's terminal port, followed by the month, day, and time of login.

which account is being used

The command is `who` with the arguments `am i` will tell you which account you are currently using. This can be useful for a system administrator if the system administrator is using the `su` command and wants to be sure about which the system administrator is currently using.

```
$ who am i
admin    ttys000  Aug 25 17:30
$
```

The command also works withut any spaces: `whoami`

```
$ whoami
admin    ttys000  Aug 25 17:30
$
```

major directories

chapter 28

summary

This chapter looks at some of the major directories (folders) on UNIX/Linux/Mac OS X.

directory listing

The following is a brief description of some of the major directories. Not all systems will have all of these directories.

/ The root directory of the entire file system.

/Applications On Mac OS X, this is the location of the Macintosh programs.

/Applications/Utilities On Mac OS X, this is the location of the graphic user interface based utility programs.

/bin A collection of binary files, better known as programs. These are the programs needed by the operating system that normal users might use. Contrast with /sbin below.

/boot or **/kernel** The files needed to boot or start your computer. Includes the bootstrap loader, the very first thing loaded when your computer starts. On Linux, it usually includes the Linux kernel in the compressed file `vmlinuz`.

/dev The devices connected to your computer (remember, everything, even hardware devices, is treated as a file in UNIX or Linux).

/etc System-wide configuration files, startup procedures, and shutdown procedures.

/home or **/users** The home directories for each user. Contains such things as user settings, customization files, documents, data, mail, and caches. The contents of this directory should be preserved during operating system updates or upgrades.

/lib or **/Library** Shared library files and kernel modules.

/lost+found Files recovered during filesystem repair.

/mnt Mount points for removable media (floppy disks, CD-ROM, Zip drives, etc.), partitions for other operating systems, network shares, and anything else temporarily mounted to the file system. Linux and UNIX don't use Windows-style drive letters.

/net Other networked systems (again, treated as files).

/opt Optional large applications and third party software. In older systems this might be **/usr/local**.

/private On Mac OS X, this is the location of the UNIX files. It is normally kept hidden from all users in Finder, but is visible in your terminal emulator and shell. Mac OS X has links to the major UNIX directories at the root level so that UNIX and Linux tools can find their files in the standard locations.

/proc A Linux-only directory. Information about processes and devices. The files are illusionary. The files don't exist on the disk, but are actually stored in memory. Many Linux utilities derive their information from these files.

/sbin System binaries (that is, system programs). These are programs that general users don't normally use (although they can). This directory is not in the PATH for general users. Contrast with /bin listed above.

/root The home directory for the system administrator (superuser or root).

/tmp Space for temporary files. Required by all forms of UNIX. This directory may automatically be cleaned on a regular basis.

/usr User binaries (programs), libraries, manuals, and documentation (docs).

/var Files that change, such as spool directories, log files, lock files, temporary files, and formatted (on use) manual pages.

Swap Virtual memory on a hard drive. Allows the memory manager to swap some data and program segments to hard drive to expand the amount of memory available beyond the limits of physical memory.

shred

chapter 29

summary

This chapter looks at `shred`, a Linux command.

`shred` is a secure method for eliminating data, overwriting a file.

```
        $ shred filename
```

Files are normally deleted by removing their file table entries. The data remains out on the hard drive (which is how file recovery programs can work).

Writing over the file actually eliminates the data so that nobody can ever read it.

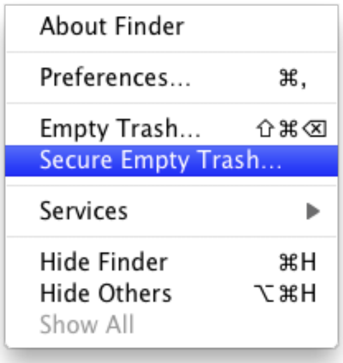
This can take a while for large files or for many files.

Files deleted with `shred` can *not* be recovered (unless you previously stored them in a backup somewhere else).

Be especially careful when using `shred` with any wildcard characters.

You should probably never run `shred` from root or superuser.

Mac OS X



Mac OS X offers secure empty trash from the graphic interface (Finder menu, Secure Empty Trash menu item, which is directly under regular Empty Trash).

Mac OS X also has the `srm` command, which is the secure equivalent of regular `rm`.

```
        $ srm -rfv -s filename
```

The `-z` option will overwrite with zeros. The `-s` option will overwrite in a single pass (one pass) of random data. The `-m` option will perform a Department of Defense seven-pass deletion. No options will result in a 35-pass file erase, which will take a long time.

df

chapter 30

summary

This chapter looks at `df`, a UNIX (and Linux) command.

`df` is used to display information on disk usage.

human readable

Use the `-h` or `-H` option to see space used in human readable format (such as kilobytes, megabytes, gigabytes, etc.). Both options do the same thing.

```
$ df -h
Filesystem      Size   Used  Avail Capacity  Mounted on
/dev/disk0s10  149Gi  117Gi   32Gi    79%    /
devfs          111Ki  111Ki    0Bi   100%    /dev
fdesc          1.0Ki  1.0Ki    0Bi   100%    /dev
map -hosts       0Bi    0Bi    0Bi   100%    /net
map auto_home    0Bi    0Bi    0Bi   100%    /home
/dev/disk1s2   498Gi   40Gi  457Gi     9%    /Volumes/msdos
/dev/disk1s1   434Gi   81Gi  353Gi    19%    /Volumes/Mac FreeAgent GoFlex Drive
$
```

type

Use the `-T` option to see the type of file system.

```
$ df -T
```

du

chapter 31

summary

This chapter looks at `du`, a UNIX (and Linux) command.

`du` is used to display usage.

display usage

Use the `-a` option to see an entry for each file in the file hierarchy.

```
$ du -a
```

Note that if you run this from root, you will get a large report of every single file on all of the mounted hard drives. best to run it from a subdirectory.

total

Use the `-c` option to see a grand total.

```
$ du -c
```

ps

chapter 32

summary

This chapter looks at `ps`, a UNIX (and Linux) command.

`ps` gives information about running processes.

info on processes

Type `ps` followed by the option `-a` (for “all”). An optional pipe to `more` or `less` will provide one page of info at a time.

```
$ ps -A |more
```

`ps` can be used to monitor the use of a server or system.

`ps` can be used to find processes that are stuck, which can then be killed.

monitor processes

`ps -ax` will provide information on most running processes. You can use the command to establish the baseline performance for your server and to monitor errant processes.

```
$ ps -ax
```

searching for specific content

You can use the following example to monitor a specific string or name from the overall output (where *`term`* is replaced with the specific string or name you want):

```
$ ps -ef|grep term
```

daemons

On Linux you use the following to get information on core processes and daemons:

```
$ /sbin/chkconfig --list
```

w

chapter 33

summary

This chapter looks at `w`, a UNIX (and Linux) command.

`w` gives information about current activity on the system, including what each user is doing.

Not to be confused with `who`, which displays users, but not the processes they are running.

info

Type `w`:

```
$ w
```

`w` does *not* provide information on background processes, which typically account for most of the server usage.

uptime

chapter 34

summary

This chapter looks at `uptime`, a UNIX (and Linux) command.

`uptime` gives information about your server's total uptime statistics.

info

Type `uptime`:

```
$ uptime
20:03  up  1:43, 2 users, load averages: 0.94 0.82 0.78
$
```

`uptime` tells you how long your server has been running. `uptime` also gives the average system load for the last few minutes. Use `uptime` for a quick performance check. Collect daily data to watch for problems from underpowered CPUs or memory management.

top

chapter 35

summary

This chapter looks at `top`, a UNIX (and Linux) command.

`top` gives the top processes (the default is by CPU usage).

using top

Type `top` and ENTER or RETURN.

```
        $ top
```

You will get updated reports (typically, once per second) on the active processes, load average, CPU usage, shared libraries, and memory usage (regions, physical, and virtual).

Type `q` to stop the `top` program.

free

chapter 36

summary

This chapter looks at `free`, a Linux command.

`free` is used to show free memory.

`free` displays the total amount of free and used physical memory and swap space in the system, as well as the buffers and cache consumed by the kernel.

size

The default is to report the memory size in bytes. Use the `-g` option to display in Gigabytes. Use the `-m` option to display in Megabytes. Use the `-k` option to display in Kilobytes. Use the `-b` option to display in Bytes.

total

Use the `-t` option to view total memory.

low/high

Use the `-l` option to view low vs. high memory usage.

count

The default is to display the information once and then return to the normal shell. Use the `-s` *n* option (where *n* is the number of times to display) to keep displaying the memory usage (until interrupted). Use the `-c` *n* or `--count=`*n* options (where *n* is the number of times to display the memory statistics).

Mac OS X

Use the Activity Monitor for this purpose. From the command line, you can get similar results from `vm_stat`. `vm_stat` is a Mach-specific version of the UNIX command `vmstat` Also:

```
$ echo -e "\n$(top -l 1 | awk '/PhysMem/';)\n"
```

vmstat

chapter 37

summary

This chapter looks at `vmstat`, a UNIX (and Linux) command.

`vmstat` is used to show virtual memory usage.

usage

Type `vmstat` to see a list of virtual memory statistics. Especially useful in a virtulaized enivronment.

```
$ vmstat
```

Mac OS X

`vm_stat` is a Mach-specific version of the UNIX command `vmstat`:

```
$ vm_stat


Mach Virtual Memory Statistics: (page size of 4096 bytes)
Pages free:                     5538.
Pages active:                 142583.
Pages inactive:                76309.
Pages wired down:              37531.
"Translation faults":        4427933.
Pages copy-on-write:           54807.
Pages zero filled:           3086159.
Pages reactivated:              6533.
Pageins:                       46357.
Pageouts:                       4453.
Object cache: 66465 hits of 76770 lookups (86% hit rate)
```

defaults

chapter 38

summary

This chapter looks at `defaults`, a Mac OS X command.

`defaults` is used to read, write, or delete defaults form a command line shell.

defaults options

Mac OS X programs use the defaults system to record user preferences and other information. Most of the information is more easily accessed through the application's Preferencess panel, but some information is normally inaccessible to the end user.

Note: Since applications do access the defaults system while they're running, you shouldn't modify the defaults of a running application.

All programs share the defaults in **NSGlobalDomain**. If the program doesn't have its own default, it uses the value from the NSGlobalDomain.

screencapture defaults

The following methods use Terminal to change the default file format and location where the screenshot is saved from the graphic user interface.

In Mac S X 10.4 (Tiger) or more recent, the default screencapture format can be changed in Terminal by using the `defaults` command. In Mac S X 10.4 (Tiger), the new default does not take effect until you logout and log back in(from the entire computer, not just from Terminal — a full restart will also work) unless you also use the `killall` command.

```
$ defaults write com.apple.screencapture type ImageFormat; killall SystemUIServer
```

The *FileFormat* can be png (Portable Network Graphic), pdf (Portable Document Format), tiff (Tagged Image File Format), jpg or jpeg (Joint Photografic Experts Group), pict (Macintosh QuickDraw Picture), bmp (Microsoft Windows Bitmap), gif (Graphics Interchange Format), psd (Adobe Photoshop Document), sgi (Silicon Graphics File Format), or tga (Truevision Targe File Format).

JPGs are saved at quality 60%.

To change the default location where the screenshot file is saved (the default is Desktop), use the following Terminal command (where *pathname* is the full path to a directory.:

```
$ defaults write com.apple.screencapture location PathName; killall SystemUIServer
```

The normal default location would be reset with the following command (where *USername* is the current account's user name.

```
$ defaults write com.apple.screencapture location /Users/UserName/Desktop; killall SystemUIServer
```

dock defaults

Use the following commands to change the 3D Dock of Mac OS X Leopard and later back to the 2D look. The `killall` restarts the Dock so that the change takes effect right away.

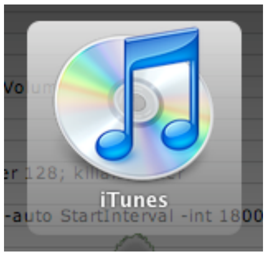
```
$ defaults write com.apple.dock no-glass -boolean YES; killall Dock
```

Return to the 3D look with the following commands:

```
$ defaults write com.apple.dock no-glass -boolean NO; killall Dock
```

Use the following commands to add a gradient behind an icon in a Dock Stack in Mac OS X:

```
$ defaults write com.apple.dock mouse-over-hilte-stack -boolean YES; killall Dock
```



Use the following commands to add a small Exposé button to the upper right of your Mac OS X screen. Clicking on the Exposé button will show all of the windows from the current application (same as the normal F10 default) and option clicking the Exposé button will show all windows from all programs (same as the normal F9 default). This will work even if you reconfigure the F9 and F10 keys to do something else.

```
$ defaults write com.apple.dock wvous-floater -bool YES; killall Dock
```

Use the following commands to remove the small Exposé button.

```
$ defaults write com.apple.dock wvous-floater -bool NO; killall Dock
```

login defaults

Use the following command to add a message to your Login window. Replace *message* with a short message. Keep the message short. As always with the `sudo` command, type very carefully.

```
$ sudo defaults write /Library/Preferences/com.apple.loginwindow LoginwindowText "message"
```

To remove the login message, type the following:

```
$ sudo defaults write /Library/Preferences/com.apple.loginwindow LoginwindowText ""
```

Mac Flashback Trojan

You can use `defaults` to determine if your Macintosh has been infected by the Mac Flashback Trojan (which enters your computer through a Java flaw).

Type the following command (copy and paste into Terminal):

```
$ defaults read /Applications/Safari.app/Contents/Info LSEnvironment
```

A clean system will report "The domain/default pair of (/Applications/Safari.app/Contents/Info, LSEnvironment) does not exist". Any other result indicates your computer is infected.

If your computer passes the first test, type the following command (copy and paste into Terminal):

```
$ defaults read ~/.MacOSX/environment DYLD_INSERT_LIBRARIES
```

A clean system will report "The domain/default pair of (/Users/*user-name*/.MacOSX/environment, DYLD_INSERT_LIBRARIES) does not exist". Any other result indicates your computer is infected.

If your computer is infected, immediately go to F-Secure at http://www.f-secure.com/v-descs/trojan-downloader_osx_flashback_i.shtml.

Downloading the latest security patches from Apple at http://support.apple.com/kb/HT5228?viewlocale=en_US&locale=en_US will help prevent infection.

init

chapter 39

summary

This chapter looks at `init`, a Linux command.

`init` is used to change the runlevel.

run levels

Linux has run levels (sometimes written as a single word, runlevel). The non-graphic mode is run level 3. the graphic mode is run level 5. Sometimes a Linux server will only boot up to the non-graphic level. You can use this command to temporarily get the graphic interface running (run level 5).

```
$ init 5
```

Edit the `/etc/inittab` file to include `id:5:initdefault:` to make a permanent change.

A chart of the Linux run levels:

RUN LEVEL	DESCRIPTION
0	System is halted
1	Single-user Mode used for system maintenance
2	Multiuser mode withut networking rarely used
3	Standard multiuser mode with networking active
4	not defined
5	Multiuser mode with graphics Like standard multiuser mode with networking active plus the X Window System graphic interface
6	Reboots the system shuts down all system services

ifconfig

chapter 40

summary

This chapter looks at `ifconfig`, a UNIX (and Linux) command.

`ifconfig` is used to configure network cards and interfaces.

view configuration

Type `ifconfig` by itself to view how your network is configured.

```
$ ifconfig
```

static IP address

Use `ifconfig` *`interface IP-address`* to set the static IP address for a particular interface

```
$ ifconfig en0 100.1.1.1
```

arp

chapter 41

summary

This chapter looks at `arp`, a UNIX (and Linux) command.

`arp` is used to display and modify the Internet-to-Ethernet address translation tables.

display all

Use the `-a` option to display all of the current ARP entries. Useful in conjunction with `ifconfig` and `route`.

```
$ arp -a
? (192.168.0.1) at 0:11:95:75:8:86 on en1 [ethernet]
? (192.168.0.106) at 0:d:60:d2:55:3f on en1 [ethernet]
? (192.168.0.255) at ff:ff:ff:ff:ff:ff on en1 [ethernet]
$
```

netstat

chapter 42

summary

This chapter looks at `netstat`, a UNIX (and Linux) command.

`netstat` is used to symbolically displays the contents of various network-related data structures.

view connections

Type `netstat` to see the list of all sockets and server connections.

```
$ netstat
```

main info

The main information for webserver administration is in the first few lines, so you can pipe to `head`:

```
$ netstat | head
```

routing addresses

Use the `-r` option to display the network routing addresses.

```
$ netstat -r
```

route

chapter 43

summary

This chapter looks at `route`, a UNIX (and Linux) command.

`route` is used to manipulate the network routing tables.

view connections

Type `route` followed by the `-v` option to list the routing tables. This is close to the same information reported by `netstat -r`.

```
        $ route -v
```

routing commands

The `route` utility can used for `add`, `flush`, `delete`, `change`, `get`, or `monitor`.

The `add` command will add a route.

The `flush` command will remove all routes.

The `delete` command will delete a specific route.

The `change` command will change aspects of a route (such as its gateway).

The `get` command will lookup and display the route for any destination.

The `monitor` command will continuously report any changes to the routing infrmation base, routing lookup misses, or suspected network partitionings.

ping

chapter 44

summary

This chapter looks at `ping`, a UNIX (and Linux) command.

`ping` is used to test a server connection.

test packets

`ping` sends test packets to a specific server to see if the server responds properly.

Type `ping` followed by `-c` *`count`* followed by an IP address or domain name.

`ping` will continue forever unless halted or you give a specific count limit.

```
$ ping -c 5 www.osdata.com
PING osdata.com (67.18.4.2): 56 data bytes
64 bytes from 192.168.0.1: icmp_seq=0 ttl=127 time=18.801 ms
64 bytes from 192.168.0.1: icmp_seq=1 ttl=127 time=9.918 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=127 time=17.216 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=127 time=6.748 ms
64 bytes from 192.168.0.1: icmp_seq=4 ttl=127 time=30.640 ms

--- 192.168.0.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 6.748/16.665/30.640/8.295 ms
$
```

`ping` will quickly either return a report or will say *destination host unreachable* on failure.

measuring

You can measure the dropped packet rate and the minimum, mean, maximum, and standard deviation of round-trip times.

When using `ping` for testing, start by pinging local host to verify that the local network interface is up and running. Then ping hosts and gateways further away.

nslookup

chapter 45

summary

This chapter looks at `nslookup`, a UNIX (and Linux) command.

`nslookup` is used to obtain domain name and IP information about a server.

information

Type `nslookup` followed by a doman name.

```
$ nslookup www.osdata.com
Server:       192.168.0.1
Address:      192.168.0.1#53

Non-authoritative answer:
www.osdata.com  canonical name = osdata.com.
Name:   osdata.com
Address: 67.18.4.2

$
```

security

You can use `nslookup` to find the DNS information for a particular host IP in your server access logs.

traceroute

chapter 46

summary

This chapter looks at `traceroute`, a UNIX (and Linux) command.

`traceroute` is used to obtain the entire path to a server or host.

entire route

Type `traceroute` followed by a doman name.

```
        $ traceroute www.osdata.com
```

You can use `traceroute` to diagnose network problems. When some people can access your website and other can’t, then `traceroute` can help you find the break or error along the network path.

etiquette

`traceroute` puts a load on every server in the path to the selected host. Therefore, only use `traceroute` when you need the diagnosis.

sysstat

chapter 47

summary

This chapter looks at `sysstat`, a server package.

package

`sysstat` is a server toolset originally created by Sebastien Godard.

Download sysstat from http://pagesperso-orange.fr/sebastien.godard/download.html.

The most famous tools ar iostat, pidstat, and sar.

`iostat` reports CPU statistics and input/output statistics for devices, partitions and network filesystems.

`mpstat` reports individual or combined processor related statistics.

`pidstar` reports statistics for Linux tasks (processes) : I/O, CPU, memory, etc.

`sar` collects, reports and saves system activity information (CPU, memory, disks, interrupts, network interfaces, TTY, kernel tables,etc.)

`sadc` is the system activity data collector, used as a backend for sar.

`sa1` collects and stores binary data in the system activity daily data file. It is a front end to sadc designed to be run from cron.

`sa2` writes a summarized daily activity report. It is a front end to sar designed to be run from cron.

`sadf` displays data collected by sar in multiple formats (CSV, XML, etc.) This is useful to load performance data into a database, or import them in a spreadsheet to make graphs.

`nfsiostat` reports input/output statistics for network filesystems (NFS).

`cifsiostat` reports CIFS statistics.

at

chapter 48

summary

This chapter looks at `at`, a UNIX (and Linux) command.

`at` is used to schedule a particular job at a particular time.

example

Type `at midnight`, followed by ENTER or RETURN.

```
        $ at midnight
```

You may see the `at>` prompt (on Mac OS X, there is no prompt).

Type a single command, followed by ENTER or RETURN.

```
        $ who > who.out
```

Type one command per line.

When finished, hold down the CONTROL key and then the D key (written ^d) to exit `at`.

```
        job 1 at Fri Jul 13 00:00:00 2012
        $
```

You will see a job number and the time it will run. This job will run all of the commands you entered.

removing an at job

Type `atrm` (for at remove), followed by the job number to remove an existing `at` job.

```
        $ atrm 1
```

timing

You can name a specific time using the YYMMDDhhmm.SS format.

You can also specify noon, midnight, or teatime (4 p.m.).

If the time has already passed, the next day is assumed.

tar

chapter 49

summary

This chapter looks at `tar`, a UNIX (and Linux) command.

Use `tar` for making backups and for retrieving open source software from the internet.

This command is used to archive directories and extract directories from archives. The `tar` archives can be used for back-up storage and for transfer to other machines.

`tar` files are often used to distribute open source programs.

`tar` is short for Tape ARchive.

create

Creating a new tar archive:

```
$ tar cvf archivename.tar dirname/
```

The c indicates create a new archive, the v indicates verbosely list file names, and the f indicates that the following is the archive file name. The v option may be left out to avoid a long list.

Creating a new gzipped tar archive:

```
$ tar cvzf archivename.tar.gz dirname/
$ tar cvzf archivename.tgz dirname/
```

The z indicates the use of gzip compression. The file extensions .tgz and .tar.gz mean the same thing and are interchangeable.

Creating a new bzip2 tar archive:

```
$ tar cvzj archivename.tar.bz2 dirname/
$ tar cvzj archivename.tbz dirname/
```

The j indicates the use of bzip2 compression. The file extensions .tbz and .tar.bz2 mean the same thing and are interchangeable.

Bzip2 takes longer to compress and decompress than gzip, but also produces a smaller compressed file.

extract

Extracting from an existing tar archive:

```
$ tar xvf archivename.tar
```

The x indicates extract files from an archive, the v indicates verbosely list file names, and the f indicates that the following is the archive file name. The v option may be left out to avoid a long list.

Extracting from a gzipped tar archive:

```
$ tar xvfz archivename.tgz
$ tar xvfz archivename.tar.gz
```

The z option indicates uncompress a gzip tar archive.

Extracting from a bzipped tar archive:

```
$ tar xvfj archivename.tbz
$ tar xvfj archivename.tar.bz2
```

The j option indicates uncompress a bzip2 tar archive.

tarpipe

It is possible to tar a set of directories (and their files and them pipe the tar to an extraction at a new location, making an exact copy of an entire set of directories (including special files) in a new location:

```
$ tar -c "$srcdir" | tar -C "$destdir" -xv
```

tarbomb

A malicious tar archive could overwrite existing files. The use of absolute paths can spread files anywhere, including overwriting existing files (even system files) anywhere. The use of parent directory references can also be used to overwrite existing files.

One can examine the tar archive without actually extracting from it.

The bsdtar program (the default tar on Mac OS X v10.6 or later) refuses to follow either absolute path or parent-directory references.

viewing

Viewing an existing tar archive (without extracting) with any of the following:

```
$ tar tvf archivename.tar
$ tar -tf archivename.tar
$ tar tf archivename.tar
$ tar --list --file=archivename.tar
```

Viewing an existing gzipped tar archive (without extracting) with any of the following:

```
$ tar tvfz archivename.tar.gz
$ tar tvfz archivename.tgz
```

Viewing an existing bzipped tar archive (without extracting) with any of the following:

```
$ tar tvfj archivename.tar.bz2
$ tar tvfj archivename.tbz
```

less

You can pipe the output of a tar file to the less command.

You can directly use the less command to open a tar archive. Less will show you the names of the files, the file sizes, the file permissions, the owner, and the group (the equivalent of `ls -l`).

```
$ less archivename.tar
```

extracting a single file

You can extract a specific single file from a large tar archive.

```
$ tar xvf archivefile.tar /complete/path/to/file
```

You can extract a specific compressed file from a large tar archive.

```
$ tar xvf archivefile.tar.gz /complete/path/to/file
$ tar xvf archivefile.tar.bz2 /complete/path/to/file
```

extracting a single directory

You can extract a specific single directory from a large tar archive.

```
$ tar xvf archivefile.tar /complete/path/to/dir/
```

You can extract a specific compressed directory from a large tar archive.

```
$ tar xvf archivefile.tar.gz /complete/path/to/dir/
$ tar xvf archivefile.tar.bz2 /complete/path/to/dir/
```

extracting a few directories

You can extract specific directories from a large tar archive by giving the name of each of the directories.

```
$ tar xvf archivefile.tar /complete/path/to/dir1/ /complete/path/to/dir2/
```

You can extract specific compressed directories from a large tar archive.

```
$ tar xvf archivefile.tar.gz /complete/path/to/dir1/ /complete/path/to/dir2/
$ tar xvf archivefile.tar.bz2 /complete/path/to/dir1/ /complete/path/to/dir2/
```

extracting a group of files

You can extract a group of files from a large tar archive by using globbing. Specify a pattern (the following example finds all .php files):

```
$ tar xvf archivefile.tar --wildcards '*.php'
```

You may use the gzip and bzip2 options as well.

adding to an existing archive

You can add an additional file to an existing tar archive with the option `-r`:

```
$ tar rvf archivefile.tar newfile
```

You can add an additional directory to an existing tar archive:

```
$ tar rvf archivefile.tar newdir/
```

This does *not* work for adding a file or directory to a compressed archive. You will get the following error message instead:

```
tar: Cannot update compressed archives
```

verifying files

You can verify a new archive file while creating is by using the `-W` option:

```
$ tar cvfW archivefile.tar dir/
```

If you see something similar to the following output, something has changed:

```
Verify 1/filename
1/filename: Mod time differs
1/filename: Size differs
```

If you just see the Verify line by itself, the file or directory is fine:

```
Verify 1/filename
```

The `-W` option does *not* work for compressed files.

For gzip compressed files, find the difference between the gzip archive and the file system:

```
$ tar dfz 1/filename.tgz
```

For bzip2 compressed files, find the difference between the bzip2 archive and the file system:

```
$ tar dfj 1/filename.tbz
```

touch

chapter 50

summary

This chapter looks at `touch`, a UNIX (and Linux) command.

Use `touch` to change the date stamp on a file.

The most common use of `touch` is to force `make` to recompile a particular file (or group of files).

how to use

The simple form of `touch` changes a named file’s timestamp to the current date and time:

```
$ touch filename
```

multiple files

Use `touch` to change all of the files in a directory:

```
$ touch *
```

specific time

You can use `touch` to set a specific time by using the `-t` option. The following command sets a file to the Solstice at the end of the Mayan calendar (11:12 a.m. December 21, 2012):

```
$ touch -t201212211112.00 filename
```

The following command sets a file to the noon on January 1, 2012:

```
$ touch -t201201011200.00 filename
```

The format is `YYYYMMDDhhmm.ss`, YYYY = four digit year, MM - two digit month, DD = two digit day, hh = two digit hour (24 hour clock), mm = two digit month, and ss = two digit second.

find

chapter 51

summary

This chapter looks at `find`, a UNIX (and Linux) command.

`find` is used to find files.

find a file

You can use `find` to quickly find a specific file. The `-x` option (this is `-xdev` on older systems) limits the search to only those directories on the existing filesystem.

```
        $ find / -name filename -type d -x
```

Note that you may want to run this command from root or superuser.

The `locate` command is more recent.

sed

chapter 52

summary

This chapter looks at `sed`, a UNIX (and Linux) command.

`sed` is a command line editor.

fixing end of line

Windows and DOS end every line with the old carriage return and line feed combination (\r\n). You cna use `sed` to convert Windows/DOS file format to UNIX file format:

```
        $ sed 's/.$//' filename
```

adding line numbers

You can use `sed` to add line numbers to all of the non-empty lines in a file:

```
        $ sed '/./=' filename | sed 'N; s/\n/ /'
```

awk

chapter 53

summary

This chapter looks at `awk`, a UNIX (and Linux) command.

`awk` is a programming language intended to serve a place between simple shell scripts and full programming languages. It is Turing complete.

remove duplicate lines

You can use `awk` to remove duplicate lines from a file.

```
        $ awk '!($0 in array) { array[$0]; print }' filename
```

screencapture

chapter 54

summary

This chapter looks at `screencaptures`, a Mac OS X-only command.

`screencapture` creates an image of the screen or a portion of the screen.

from the graphic user interface

The normal method for obtaining a screen capture is through the graphic user interface. Command-Shift-3 takes a screenshot of the screen and saves it as a file to the desktop under the name of "Picture 1" (or next available number if there are already screenshots saved there).

If you have multiple monitors connected, each monitor is saved as a separate picture, named "Picture 1", "Picture 1(2)", "Picture 1(3)", etc.

With Mac OS X 10.6 (Snow Leopard) the default name changes to "Screen shot YYYY-MM-DD at HH.MM.SS XM", where YYY=year, MM=month, DD=day, HH=hour, MM=minute, SS=second, and XM = either AM or PM.

The basic screen capture options:

- **Command-Shift-3:** Take a screenshot of the entire screen and save it as a file on the desktop.
- **Command-Shift-4, then select an area:** Take a screenshot of an area and save it as a file on the desktop.
- **Command-Shift-4, then space, then click on a window:** Take a screenshot of a selected window and save it as a file on the desktop.
- **Command-Control-Shift-3:** Take a screenshot of the screen and save it to the clipboard.
- **Command-Control-Shift-4, then select an area:** Take a screenshot of an area and save it to the clipboard.
- **Command-Control-Shift-4, then space, then click on a window:** Take a screenshot of a selected window and save it to the clipboard.

In Mac OS X 5 (Leopard) or more recent, the following keys can be held down when selecting an area (with either Command-Shift-4 or Command-Control-Shift-4):

- **Space:** Used to lock the size of the selected region and move the selected region as the mouse moves.
- **Shift:** Used to resize only one edge of the slected region.
- **Option:** Used to resize the selected region with its center as the anchor point.

Different versions of Mac OS X have different default file formats for saving the screenshot:

- **Mac OS X 10.2 (Jaguar):** jpg
- **Mac OS X 10.3 (Panther):** pdf
- **Mac OS X 10.4 (Tiger):** png

changing defaults

The following methods use Terminal to change the default file format and location where the screenshot is saved from the graphic user interface.

In Mac S X 10.4 (Tiger) or more recent, the default screencapture format can be changed in Terminal by using the `defaults` command. In Mac S X 10.4 (Tiger), the new default does not take effect until you logout and log back in(from the entire computer, not just from Terminal — a full restart will also work) unless you also use the `killall` command.

```
$ defaults write com.apple.screencapture type ImageFormat
$ killall SystemUIServer
```

The *FileFormat* can be png (Portable Network Graphic), pdf (Portable Document Format), tiff (Tagged Image File Format), jpg or jpeg (Joint Photographic Experts Group), pict (Macintosh QuickDraw Picture), bmp (Microsoft Windows Bitmap), gif (Graphics Interchange Format), psd (Adobe Photoshop Document), sgi (Silicon Graphics File Format), or tga (Truevision Targe File Format).

JPGs are saved at quality 60%.

To change the default location where the screenshot file is saved (the default is Desktop), use the following Terminal command (where *pathname* is the full path to a directory.:

```
$ defaults write com.apple.screencapture location PathName
$ killall SystemUIServer
```

The normal default location would be reset with the following command (where *USername* is the current account's user name.

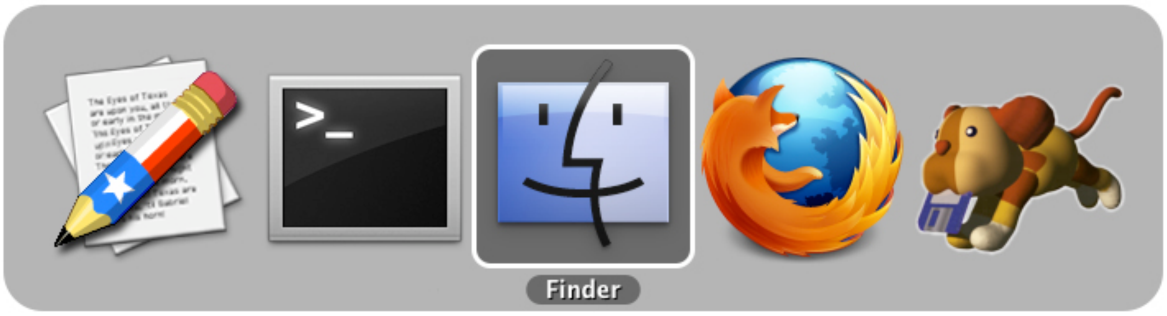
```
$ defaults write com.apple.screencapture location /Users/UserName/Desktop
$ killall SystemUIServer
```

command line screenshots

You can also take screenshots from Terminal.

I needed a screenshot of the CONTROL-TAB selection of a program, but in the graphic user interface, I couldn't simultaneously run Command-Tab and Command-Shift-4, so I used the following command in Terminal to set a 10 second delay and save the screenshot selection:

```
$ screencapture -T 10 -t png controltab.png
```



You can add this command to your Mac OS X scripts.

The format is `screencapture` *`options filenames`*. List more than one file name if you have more than one monitor. You can use the options in any combination.

You can use the *filename* to change the file name from the normal default and to set a relative path to a directory/folder of your choice.

```
$ screencapture [-icMPmwsWxSCUt] [files]
```

The basic use, which takes an immediate screenshot in the default format and stores it with the designated filename (in this case "Picture1") in the user's home directory (not the desktop).

```
$ screencapture Picture1
```

Force the screenshot to go to the clipboard (the equivalent of the Command-Shift-Control- choices).

```
$ screencapture -c [files]
```

Capture the cursor as well as the screen. This applies only in non-interactive modes (such as a script).

```
$ screencapture -C [files]
```

Display errors to the user graphically.

```
$ screencapture -d [files]
```

Capture the screenshot interactively by either selection or window (the equivalent of Command-Shift-4). Use the CONTROL key to cause the screenshot to go to the clipboard. Use the SPACE key to toggle between mouse selection and window selection modes. Use the ESCAPE key to cancel the interactive screen shot.

```
$ screencapture -i [file]
```

Use the `-m` option to only capture the main monitor. This does not work if the `-i` option is also set.

```
$ screencapture -m [file]
```

Send the screenshot to a new Mail message.

```
$ screencapture -M [files]
```

Use the `-o` option in window capture mode to only capture the window and to *not* capture the shadow of the window.

```
$ screencapture - o [file]
```

After savng the screenshot, open the screen capture output in Preview.

```
$ screencapture -P [files]
```

Use `-s` to only allow mouse selection mode.

```
$ screencapture -s [files]
```

Use `-w` to only allow window selection mode.

```
$ screencapture -w [file]
```

Use `-W` to start interaction in the window selection mode.

```
$ screencapture -W [file]
```

Use the `-S` option in window capture mode to capture the screen rather than the window.

```
$ screencapture -S [files]
```

Set the format with the `-t` option. The *Format* can be png (Portable Network Graphic), pdf (Portable Document Format), tiff (Tagged Image File Format), jpg or jpeg (Joint Photographic Experts Group), pict (Macintosh QuickDraw Picture), bmp (Microsoft Windows Bitmap), gif (Graphics Interchange Format), psd (Adobe Photoshop Document), sgi (Silicon Graphics File Format), or tga (Truevision Targe File Format)

```
$ screencapture -tFormat [files]
```

Set a delay time in seconds. The default is five seconds.

```
$ screencapture -TSeconds [files]
```

Prevent the playing of sounds (no camera click sound).

```
$ screencapture -x [files]
```

installing software from source code

chapter 55

summary

This chapter looks at installing software from source code.

While it is possible to do a lot with just preinstalled software and precompiled software, those working with large systems will sooner or later need to download the raw source code and use that source code to create their own copy of the latest version of some software on their servers or other computer systems.

gcc

The vast majority of open source software is written in the C programming language. The GNU foundation C compiler, known as `gcc`, is the most commonly used C compiler.

Apple’s Mac OS X Developer Tools include a copy of `gcc`, but also include an Apple specific variation that includes specialized features useful for development of Macintosh, iPhone, and iPad software.

Xcode

Xcode is an Apple integrated development environment. It includes the Instruments analysis tool, iOS Simulator (useful for testing your software on the Mac before it is ready for deployment to real devices), and the latest Mac OS X and iOS SDKs.

Fink

Fink (German for the bird finch) is an open source program that automates the process of downloading, configuring, compiling, and installing hundreds of software packages from the original source code.

manual installation

There are two major methods for manually installing software from source: the standard method (used for most Linux software) and the Perl method (used for Perl modules).

test bed

chapter 56

summary

This chapter looks at a test bed for writing this book.

At the time of this writing, this test bed does not yet exist. The author was asked to put together a list of test bed requirements, emphasizing saving money as much as possible. Rather than keep this test bed as a secret, it seems that it would be a good idea to share this information with the readers. It also makes sense to share with the reader the process of installing each component of the test bed.

stages

There are five basic stages of deployment of the test bed.

1. **initial main development computer** This is a low cost used computer to get work started, as well as the additional essential items to support that development.
2. **initial server machines** These are the first and most important machines for testing server deployment.
3. **mobile computing** These are the items needed for creating and testing mobile apps.
4. **additional servers** These are the additional desired test platforms for server and desktop testing.
5. **content creation** These are the items needed for full content creation. These will be divided into additional stages.

initial main development computer

This is a low cost used computer to get work started, as well as the additional essential items to support that development.

The initial main development computer is older technology chosen to greatly lower costs, yet still provide a useful platform for development.

Hardware: Note that this is available only as a used computer — Apple Power Mac G5 “Cipher”:

- **Processor:** DP DC “Quadcore“ 2.5 GHz Power PC 970MP (G5)
- **Cooling:** Panasonic LCS
- **Graphics:** GeForce 7800 GT with 512 MB of DDR RAM
- **Memory:** 16GB of 533 MHz PC2-4200 DDR2 SDRAM
- **Hard Drive:** Two of 500GB 7200 rpm drives
- **Wireless:** AirPort Extreme with Bluetooth 2.0+EDR combo card
- **Optical Drives:** Two of 16x SuperDrive (DVD+R DL/DVD±RW/CD-RW)
- **OS:** Mac OS X 10.5.8 Leopard
- **Monitor:** Apple Cinema Display (30-inch DVI)
- **Input:**Apple extended keyboard and Mighty Mouse

Software: Note that most of this software is only available used.

- Tex-Edit Plus (new-shareware)
- Fetch (new)
- Apple Developer Tools (download free from Apple)
- Adobe Creative Suite 4 Design Premium, including Adobe InDesign CS4, Photoshop CS4 Extended, Illustrator CS4, Flash CS4 Professional, Dreamweaver CS4, Fireworks CS4, and Acrobat 9 Pro with additional tools and services
- Apple Final Cut Pro 5
- Apple Logic Pro 8
- Fink (free download)

initial server machines

These are the first and most important machines for testing server deployment.

FreeBSD computer

A computer set up with FreeBSD.

Linux Mint computer

A computer set up with Linux Mint.

OpenVMS computer

An itanium computer set up with a hobbyist licensed copy of OpenVMS.

Windows computer

Need to find someone willing to work in the horrid Windows environment who can check that the software works correctly in that environment. Microsoft's monopolistic approach may prevent deployment of system-independent software.

mobile computing

These are the items needed for creating and testing mobile apps.

iPhnne

An iPhone and development system.

Android

An Android and development system.

iPad

An iPad and development system.

BlackBerry

A BlackBerry and development system.

Windows Phone

A Windows Phone 7 and development system.

Nokia

A Nokia and development system.

additional servers

These are the additional desired test platforms for server and desktop testing. There is no particular order implied at this time, other than getting to Red hat Enterprise, Ubuntu, and Solaris as soon as possible. We also want an IBM Power or Z system (even if it is an old one) with FORTRAN, PL/I, COBOL, and C compilers, as well as IBM's server and system administration software.

Red Hat Enterprise Linux computer

A computer set up with Red Hat Enterprise Linux.

Ubuntu Linux computer

A computer set up with Ubuntu Linux.

Solaris computer

An UltraSPARC computer set up with Solaris and Oracle Database.

Fedora Linux computer

A computer set up with Fedora Linux.

OpenSUSE Linux computer

A computer set up with OpenSUSE Linux.

CentOS Linux computer

A computer set up with CentOS Linux.

Oracle Linux computer

A computer set up with Oracle Linux and Oracle Database.

Debian Linux computer

A computer set up with Debian Linux.

IBM zEneterprise EC12 computer

An IBM zEnterprise EC12 model HA1 computer running z/OS with Enterprise PL/I, FORTRAN, Enterprise COBOL, and XL C/C++ compilers and APL interpreter. The HA1 with 101 processor units in 4 books, which include Central Processor (CP), Internal Coupling facility (ICF), Integrated facility for Linux (IFL), Additional System Assist Processor (SAP), System z Application Assist Processor (zAAP), and System z Intergrated Information Processor (zIIP) — the exact mix of the 101 processors to be determined. Water cooled option. 3040 GB of memory. zBX Model 003 attachment (with IBM POWER7 blade running PowerVM Enterprise Edition, IBM POWER7 blade running AIX, IBM BladeCenter HX5 blade running Red Hat Eterprise Linux, and IBM BladeCenter HX5 blade running SUSE Linux Enterprise Server, up to 112 blades). Trusted Key Entry (TKE) workstation. IBM Crypto Express 4S digital signature cryptography and EAL 5+ certification. IBM zAware. IBM DB2 Analytics Accelerator. IBM SmartCloud Enterprise+ for System z. IBM System Storage DS8000. IBM TS1140 tape drives with encryption.

The zEnterprise EC12 is available in five hardware models: H20, H43, H66, H89, HA1A1. This includes the capability to support over 100 configurable cores.

content creation

These are the items needed for full content creation. These will be divided into additional stages.

history of computers

appendix A

Notes on the summary at the end of each year:

Year Year that items were introduced.

Operating Systems Operating systems introduced in that year.

Programming Languages Programming languages introduced. While only a few programming languages are appropriate for operating system work (such as Ada, BLISS, C, FORTRAN, and PL/I, the programming languages available with an operating system greatly influence the kinds of application programs available for an operating system.

Computers Computers and processors introduced. While a few operating systems run on a wide variety of computers (such as UNIX and Linux), most operating systems are closely or even intimately tied to their primary computer hardware. Speed listings in parenthesis are in operations per second (OPS), floating point operatins per second (FLOPS), or clock speed (Hz).

Software Software programs introduced. Some major application programs that became available. Often the choice of operating system and computer was made by the need for specific programs or kinds of programs.

Games Games introduced. It may seem strange to include games in the time line, but many of the advances in computer hardware and software technologies first appeared in games. As one famous example, the roots of UNIX were the porting of an early computer game to new hardware.

Technology Major technology advances.

antiquity

The earliest calculating machine was the abacus, believed to have been invented in Babylon around 2400 BCE. The abacus was used by many different cultures and civilizations, including the major advance known as the Chinese abacus from the 2nd Century BCE.

200s BCE

The Antikythera mechanism, discovered in a shipwreck in 1900, is an early mechanical analog computer from between 150 BCE and 100 BCE. The Antikythera mechanism used a system of 37 gears to compute the positions of the sun and the moon through the zodiac on the Egyptian calendar, and possibly also the fixed stars and five planets known in antiquity (Mercury, Venus, Mars, Jupiter, and Saturn) for any time in the future or past. The system of gears added and subtracted angular velocities to compute differentials. The Antikythera mechanism could accurately predict eclipses and could draw up accurate astrological charts for important leaders. It is likely that the Antikythera mechanism was based on an astrological computer created by Archimedes of Syracuse in the 3rd century BCE.

100s BCE

The Chinese developed the South Pointing Chariot in 115 BCE. This device featured a differential gear, later used in modern times to make analog computers in the mid-20th Century.

400s

The Indian grammarian Panini wrote the *Ashtadhyayi* in the 5th Century BCE. In this work he created 3,959 rules of grammar for India's Sanskrit language. This important work is the oldest surviving linguistic book and introduced the idea of metarules, transformations, and recursions, all of which have important applications in computer science.

1400s

The Inca created digital computers using giant loom-like wooden structures that tied and untied knots in rope. The knots were digital bits. These computers allowed the central government to keep track of the agricultural and economic details of their far-flung empire. The Spanish conquered the Inca during fighting that stretched from 1532 to 1572. The Spanish destroyed all but one of the Inca computers in the belief that the only way the machines could provide the detailed information was if they were Satanic divination devices. Archaeologists have long known that the Inca used knotted strings woven from cotton, llama wool, or alpaca wool called khipu or quipus to record accounting and census information, and possibly calendar and astronomical data and literature. In recent years archaeologists have figured out that the one remaining device, although in ruins, was clearly a computer.

1800s

Charles Babbage created the difference engine and the analytical engine, often considered to be the first modern computers. Augusta Ada King, the Countess of Lovelace, was the first modern computer programmer.

In the 1800s, the first computers were programmable devices for controlling the weaving machines in the factories of the Industrial Revolution. Created by Charles Babbage, these early computers used Punch cards as data storage (the cards contained the control codes for the various patterns). These cards were very similiar to the famous Hollerinth cards developed later. The first computer programmer was Lady Ada, for whom the Ada programming language is named.

In 1822 Charles Babbage proposed a difference engine for automated calculating. In 1933 Babbage started work on his Analytical Engine, a mechanical computer with all of the elements of a modern computer, including control, arithmetic, and memory, but the technology of the day couldn't produce gears with enough precision or reliability to make his computer possible. The Analytical Engine would have been programmed with Jacquard's punched cards. Babbage designed the Difference Engine No.2. Lady Ada Lovelace wrote a program for the Analytical Engine that would have correctly calculated a sequence of Bernoulli numbers, but was never able to test her program because the machine wasn't built.

George Boole introduced what is now called Boolean algebra in 1854. This branch of mathematics was essential for creating the complex circuits in modern electronic digital computers.

1900s

In the 1900s, researchers started experimenting with both analog and digital computers using vacuum tubes. Some of the most successful early computers were analog computers, capable of performing advanced calculus problems rather quickly. But the real future of computing was digital rather than analog. Building on the technology and math used for telephone and telegraph switching networks, researchers started building the first electronic digital computers.

Leonardo Torres y Quevedo first proposed floating point arithmetic in Madrid in 1914. Konrad Zuse independently proposed the same idea in Berlin in 1936 and built it into the hardware of his Zuse computer. George Stibitz also indpendently proposed the idea in New Jersey in 1939.

The first modern computer was the German Zuse computer (Z3) in 1941. In 1944 Howard Aiken of Harvard University created the Harvard Mark I and Mark II. The Mark I was primarily mechanical, while the Mark II was primarily based on reed relays. Telephone and telegraph companies had been using reed relays for the logic circuits needed for large scale switching networks.

The first modern electronic computer was the ENIAC in 1946, using 18,000 vacuum tubes. See below for information on Von Neumann's important contributions.

The first solid-state (or transistor) computer was the TRADIC, built at Bell Laboratories in 1954. The transistor had previously been invented at Bell Labs in 1948.

1938

Computers: Zuse Z1 (Germany, 1 OPS, first mechanical programmable binary computer, storage for a total of 64 numbers stored as 22 bit floating point numbers with 7-bit exponent, 15-bit signifocana [one implicit bit], and sign bit); Konrad Zuse called his floating point hardware "semi-logarithmic notation" and included the ability to handle infinity and undefined.

1941

The first modern computer was the German Zuse computer (Z3) in 1941.

Computers: Atanasoff-Berry Computer; Zuse Z3 (Germany, 20 OPS, added floating point exceptions, plus and minus infinity, and undefined)

1942

Computers: work started on Zuse Z4

1943

In 1943 Howard Aiken of Harvard University created the Harvard Mark I. The Mark I was primarily mechanical.

Computers: Harvard Mark I (U.S.); Colossus 1 (U.K., 5 kOPS)

1944

In 1944 Howard Aiken of Harvard University created the Harvard Mark II. While the Mark I was primarily mechanical, the Mark II was primarily based on reed relays. Telephone and telegraph companies had been using reed relays for the logic circuits needed for large scale switching networks.

Computers: Colossus 2 (U.K., single processor, 25 kOPS); Harvard Mark II and AT&T Bell Labortories' Model V (both relay computers) were the first American computers to include floating point hardware

1945

Plankalkül (Plan Calculus), created by Konrad Zuse for the Z3 computer in Nazi germany, may have been the first programming language (other than assemblers). This was a surprisingly advanced programming language, with many features that didn't appear again until the 1980s.

Programming Languages: Plankalkül
Computers: Zuse Z4 (relay based computer, first commercial computer)

von Neumann architecture

John Louis von Neumann, mathematician (born János von Neumann 28 December 1903 in Budapest, Hungary, died 8 February 1957 in Washington, D.C.), proposed the *stored program concept* while professor of mathemtics (one of the orginal six) at Princeton University's Institute for Advanced Services, in which programs (code) are stored in the same memory as data. The computer knows the difference between code and data by which it is attempting to access at any given moment. When evaluating code, the binary numbers are decoded by some kind of physical logic circuits (later other methods, such as microprogramming, were introduced), and then the instructions are run in hardware. This design is called **von Neumann architecture** and has been used in almost every digital computer ever made.

Von Neumann architecture introduced flexibility to computers. Previous computers had their programming hard wired into the computer. A particular computer could only do one task (at the time, mostly building artillery tables) and had to be physically rewired to do any new task.

By using numeric codes, von Neumann computers could be reprogrammed for a wide variety of problems, with the decode logic remaining the same.

As processors (especially super computers) get ever faster, the *von Neumann bottleneck* is starting to become an issue. With data and code both being accessed over the same circuit lines, the processor has to wait for one while the other is being fetched (or written). Well designed data and code caches help, but only when the requested access is already loaded into cache. Some researchers are now experimenting with **Harvard architecture** to solve the von Neumann bottleneck. In Harvard arrchitecture, named for Howard Aiken's experimental Harvard Mark I (ASCC) calculator [computer] at Harvard University, a second set of data and address lines along with a second set of memory are set aside for executable code, removing part of the conflict with memory accesses for data.

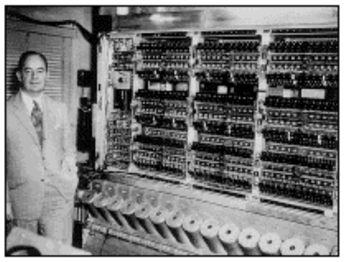
Von Neumann became an American citizen in 1933 to be eligible to help on top secret work during World War II. There is a story that Oskar Morganstern coached von Neumann and Kurt Gödel on the U.S. Constitution and American history while driving them to their immigration interview. Morganstern asked if they had any questions, and Gödel replied that he had no questions, but had found some logical inconsistencies in the Constitution that he wanted to ask the Immigration officers about. Morganstern recommended that he not ask questions, but just answer them.

Von Neumann occassionally worked with Alan Turing in 1936 through 1938 when Turing was a graduate student at Princeton. Von Neumann was exposed to the concepts of logical design and universal machine proposed in Turing's 1934 paper "On Computable Numbers with an Application to the Entschiedungs-problem".

Von Neumann worked with such early computers as the Harvard Mark I, ENIAC, EDVAC, and his own IAS computer.

Early research into computers involved doing the computations to create tables, especially artillery firing tables. Von Neumann was convinced that the future of computers involved applied mathematics to solve specific problems rather than mere table generation. Von Neumann was the first person to use computers for mathematical physics and economics, proving the utility of a general purpose computer.

Von Neumann proposed the concept of stored programs in the 1945 paper "First Draft of a Report on the EDVAC". Influenced by the idea,



Maurice Wilkes of the Cambridge University Mathematical Laboratory designed and built the EDSAC, the world’s first operational, production, stored-program computer.

The first stored computer program ran on the Manchester Mark I [computer] on June 21, 1948.

Von Neumann foresaw the advantages of parallelism in computers, but because of construction limitations of the time, he worked on sequential systems.

Von Neumann advocated the adoption of the bit as the measurement of computer memory and solved many of the problems regarding obtaining reliable answers from unreliable computer components.

Interestingly, von Neumann was opposed to the idea of compilers. When shown the idea for FORTRAN in 1954, von Neumann asked “Why would you want more than machine language?”. Von Neumann had graduate students hand assemble programs into binary code for the IAS machine. Donald Gillies, a student at Princeton, created an assembler to do the work. Von Neumann was angry, claiming “It is a waste of a valuable scientific computing instrument to use it to do clerical work”.

Von Neumann also did important work in set theory (including measure theory), the mathematical foundation for quantum theory (including statistical mechanics), self-adjoint algebras of bounded linear operators on a Hilbert space closed in weak operator topology, non-linear partial differential equations, and automata theory (later applied to cmputers). His work in economics included his 1937 paper “A Model of General Economic Equilibrium” on a multi-sectoral growth model and his 1944 book “Theory of Games and Economic Behavior” (co-authored with Morgenstern) on game theory and uncertainty.

I leave the discussion of von Neumann with a couple of quotations:

> “If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is.”

> “Anyone who considers arithmetical methods of producing random numbers is, of course, in a state of sin.”

bare hardware

In the earliest days of electronic digital computing, everything was done on the bare hardware. Very few computers existed and those that did exist were experimental in nature. The researchers who were making the first computers were also the programmers and the users. They worked directly on the “bare hardware”. There was no operating system. The experimenters wrote their programs in machine or assembly language and a running program had complete control of the entire computer. Often programs and data were entered by hand through the use of toggle switches. Memory locations (both data and programs) could be read by viewing a series of lights (one for each binary digit). Debugging consisted of a combination of fixing both the software and hardware, rewriting the object code and changing the actual computer itself.

The lack of any operating system meant that only one person could use a computer at a time. Even in the research lab, there were many researchers competing for limited computing time. The first solution was a reservation system, with researchers signing up for specific time slots. The earliest billing systems charged for the entire computer and all of its resources (regardless of whether used or not) and was based on outside clock time, being billed from the scheduled start to scheduled end times.

The high cost of early computers meant that it was essential that the rare computers be used as efficiently as possible. The reservation system was not particularly efficient. If a researcher finished work early, the computer sat idle until the next time slot. If the researcher’s time ran out, the researcher might have to pack up his or her work in an incomplete state at an awkward moment to make room for the next researcher. Even when things were going well, a lot of the time the computer actually sat idle while the researcher studied the results (or studied memory of a crashed program to figure out what went wrong). Simply loading the programs and data took up some of the scheduled time.

computer operators

One solution to this problem was to have programmers prepare their work off-line on some input medium (often on punched cards, paper tape, or magnetic tape) and then hand the work to a computer operator. The computer operator would load up jobs in the order received (with priority overrides based on politics and other factors). Each job still ran one at a time with complete control of the computer, but as soon as a job finished, the operator would transfer the results to some output medium (punched tape, paper tape, magnetic tape, or printed paper) and deliver the results to the appropriate programmer. If the program ran to completion, the result would be some end data. If the program crashed, memory would be transferred to some output medium for the programmer to study (because some of the early business computing systems used magnetic core memory, these became known as “core dumps”).

The concept of computer operators dominated the mainframe era and continues today in large scale operations with large numbers of servers.

device drivers and library functions

Soon after the first successes with digital computer experiments, computers moved out of the lab and into practical use. The first practical application of these experimental digital computers was the generation of artillery tables for the British and American armies. Much of the early research in computers was paid for by the British and American militaries. Business and scientific applications followed.

As computer use increased, programmers noticed that they were duplicating the same efforts.

Every programmer was writing his or her own routines for I/O, such as reading input from a magnetic tape or writing output to a line printer. It made sense to write a common device driver for each input or putput device and then have every programmer share the same device drivers rather than each programmer writing his or her own. Some programmers resisted the use of common device drivers in the belief that they could write “more efficient” or faster or "“better” device drivers of their own.

Additionally each programmer was writing his or her own routines for fairly common and repeated functionality, such as mathematics or string functions. Again, it made sense to share the work instead of everyone repeatedly “reinventing the wheel”. These shared functions would be organized into libraries and could be inserted into programs as needed. In the spirit of cooperation among early researchers, these library functions were published and distributed for free, an early example of the power of the open source approach to software development.

Computer manufacturers started to ship a standard library of device drivers and utility routines with their computers. These libraries were often called a **runtime library** because programs connected up to the routines in the library at run time (while the program was running) rather than being compiled as part of the program. The commercialization of code libraries ended the widespread free sharing of software.

Manufacturers were pressured to add security to their I/O libraries in order to prevent tampering or loss of data.

input output control systems

The first programs directly controlled all of the computer’s resources, including input and output devices. Each individual program had to include code to control and operate each and every input and/or output device used.

One of the first consolidations was placing common input/output (I/O) routines into a common library that could be shared by all programmers. I/O was separated from processing.

These first rudimentary operating systems were called an Input Output Control System or IOCS.

Computers remained single user devices, with main memory divided into an IOCS and a user section. The user section consisted of program, data, and unused memory.

The user remained responsible for both set up and tear down.

Set up included loading data and program, by front panel switches, punched card, magnetic tapes, paper tapes, disk packs, drum drives, and other early I/O and storage devices. Paper might be loaded into printers, blank cards into card punch mahcines, and blank or formatted tape into tape drives, or other output devices readied.

Tear down would include unmounting tapes, drives, and other media.

The very expensive early computers sat idle during both set up and tear down.

This waste led to the introduction of less expensive I/O computers. While one I/O computer was being set up or torn down, another I/O computer could be communicating a readied job with the main computer.

Some installations might have several different I/O computers connected to a single main computer to keep the expensive main computer in use. This led to the concept of multiple I/O channels.

monitors

As computers spread from the research labs and military uses into the business world, the accountants wanted to keep more accurate counts of time than mere wall clock time.

This led to the concept of the **monitor**. Routines were added to record the start and end times of work using computer clock time. Routines were added to I/O library to keep track of which devices were used and for how long.

With the development of the Input Output Control System, these time keeping routines were centralized.

You will notice that the word monitor appears in the name of some operating systems, such as FORTRAN Monitor System. Even decades later many programmers still refer to the operating system as the monitor.

An important motivation for the creation of a monitor was more accurate billing. The monitor could keep track of actual use of I/O devices and record runtime rather than clock time.

For accurate time keeping the monitor had to keep track of when a program stopped running, regardless of whether it was a normal end of the program or some kind of abnormal termination (such as aa crash).

The monitor reported the end of a program run or error conditions to a computer operator, who could load the next job waiting, rerun a job, or take other actions. The monitor also notified the computer operator of the need to load or unload various I/O devices (such as changing tapes, loading paper into the printer, etc.).

1946

The first modern electronic computer was the ENIAC in 1946, using 18,000 vacuum tubes.

Computers: UPenn Eniac (5 kOPS); Colossus 2 (parallel processor, 50 kOPS)
Technology: electrostatic memory

1948

Computers: IBM SSEC; Manchester SSEM
Technology: random access memory; magnetic drums; transistor

1949

Short Code created in 1949. This programming language was compiled into machine code by hand.

Programming Languages: Short Code
Computers: Manchester Mark 1
Technology: registers

1950s

Some operating systems from the 1950s include: FORTRAN Monitor System, General Motors Operating System, Input Output System, SAGE, and SOS.

SAGE (Semi-Automatic Ground Environment), designed to monitor weapons systems, was the first real time control system.

1951

Grace Hopper starts work on A-0.

Computers: Ferranti Mark 1 (first commercial computer); Leo I (frst business computer); UNIVAC I, Whirlwind

1952

Autocode, a symbolic assembler for the Manchester Mark I computer, was created in 1952 by Alick E. Glennie. Later used on other computers.

A-0 (also known as AT-3), the first compiler, was created in 1952 by Grace Murray Hopper. She later created A-2, ARITH-MATIC, MATH-MATIC, and FLOW-MATIC, as well as being one of the leaders in the development of COBOL.Grace Hopper was working for Remington Rand at the time. Rand released the language as MATH-MATIC in 1957.

According to some sources, first work started on FORTRAN.

Programming Languages: Autocode; A-0; FORTRAN
Computers: UNIVAC 1101; IBM 701
Games: OXO (a graphic version of Tic-Tac-Toe created by A.S. Douglas on the EDSAC computer at the University of Cambridge to demonstrate ideas on human-computer interaction)

1953

Computers: Strela

1954

The first solid-state (or transistor) computer was the TRADIC, built at Bell Laboratories in 1954. The transistor had previously been invented at Bell Labs in 1948.

FORTRAN (FORmula TRANslator) was created in 1954 by John Backus and other researchers at International Business Machines (now IBM). Released in 1957. FORTRAN is the oldest programming language still in common use. Identifiers were limited to six characters. Elegant representation of mathematic expressions, as well as relatively easy input and output. FORTRAN was based on A-0.

> "Often referred to as a *scientific language,* FORTRAN was the first *high-level* language, using the first compiler ever developed. Prior to the development of FORTRAN computer programmers were required to program in machine/assembly code, which was an extremely difficult and time consuming task, not to mention the dreadful chore of debugging the code. The objective during its design was to create a programming language that would be: simple to learn, suitable for a wide variety of applications, *machine independent,* and would allow complex mathematical expressions to be stated similarly to regular algebraic notation. While still being almost as efficient in execution as assembly language. Since FORTRAN was so much easier to code, programmers were able to write programs 500% faster than before, while execution efficiency was only reduced by 20%, this allowed them to focus more on the problem solving aspects of a problem, and less on coding.
>
> "FORTRAN was so innovative not only because it was the first high-level language [still in use], but also because of its compiler, which is credited as giving rise to the branch of computer science now known as *compiler theory*. Several years after its release FORTRAN had developed many different *dialects,* (due to special *tweaking* by programmers trying to make it better suit their personal needs) making it very difficult to transfer programs from one machine to another." —Neal Ziring, The Language Guide, University of Michigan
>
> "Some of the more significant features of the language are listed below:" —Neal Ziring, The Language Guide, University of Michigan
>
> - **Simple to learn** - when FORTRAN was design one of the objectives was to write a language that was easy to learn and understand.
> - **Machine Independent** - allows for easy transportation of a program from one machine to another.
> - **More natural ways to express mathematical functions** - FORTRAN permits even severely complex mathematical functions to be expressed similarly to regular algebraic notation.
> - **Problem orientated language**
> - **Remains close to and exploits the available hardware**
> - **Efficient execution** - there is only an approximate 20% decrease in efficiency as compared to assembly/machine code.
> - **Ability to control storage allocation** -programmers were able to easily control the allocation of storage (although this is considered to be a dangerous practice today, it was quite important some time ago due to limited memory.
> - **More freedom in code layout** - unlike assembly/machine language, code does not need to be laid out in rigidly defined columns, (though it still must remain within the parameters of the FORTRAN source code form).
>
> "42. You can measure a programmer's perspective by noting his attitude on the continuing vitality of FORTRAN." —Alan Perlis, *Epigrams on Programming,* ACM's SIGPLAN Notices Volume 17, No. 9, September 1982, pages 7-13

Programming Languages: FORTRAN
Computers: IBM 650; IBM 704 (vacuum tube computer with floating point); IBM NORC (67 kOPS)
Technology: magnetic core memory

1955

Operating Systems: GMOS (General Motors OS for IBM 701)
Computers: Harwell CADET

batch systems

Batch systems automated the early approach of having human operators load one program at a time. Instead of having a human operator load each program, software handled the scheduling of jobs. In addition to programmers submitting their jobs, end users could submit requests to run specific programs with specific data sets (usually stored in files or on cards). The operating system would schedule "batches" of related jobs. Output (punched cards, magnetic tapes, printed material, etc.) would be returned to each user.

General Motors Operating System, created by General Motors Research Laboratories in early 1956 (or late 1955) for thieir IBM 701 mainframe is generally considered to be the first batch operating system and possibly the first “real” operating system.

The operating system would read in a program and its data, run that program to completion (including outputing data), and then load the next program in series as long as there were additional jobs available.

Batch operating systems used a Job Control Language (JCL) to give the operating system instructions. These instructions included designation of which punched cards were data and which were programs, indications of which compiler to use, which centralized utilities were to be run, which I/O devices might be used, estimates of expected run time, and other details.

This type of batch operating system was known as a single stream batch processing system.

Examples of operating systems that were primarily batch-oriented include: BKY, BOS/360, BPS/360, CAL, and Chios.

1956

Researchers at MIT begin experimenting with direct keyboard input into computers.

IPL (Information Processing Language) was created in 1956 by A. Newell, H. Simon, and J.C. Shaw. IPL was a low level list processing language which implemented recursive programming.

Programming Languages: IPL
Operating Systems: GM-NAA I/O
Computers: IBM 305 RAMAC; MIT TX-0 (83 kOPS)
Technology: hard disk

1957

MATH-MATIC was released by the Rand Corporation in 1957. The language was derived from Grace Murray Hopper’s A-0.

FLOW-MATIC, also called B-0, was created in 1957 by Grace Murray Hopper.

The first commercial FORTRAN program was run at Westinghouse. The first compile run produced a missing comma diagnostic. The second attempt was a success.

The U.S. government created the Advanced Research Project Group (ARPA) in esponse to the Soviet Union’s launching of Sputnik. ARPA was intended to develop key technology that was too risky for private business to develop.

Programming Languages: FLOW-MATIC; MATH-MATIC
Computers: IBM 608
Technology: dot matrix printer

1958

FORTRAN II in 1958 introduces subroutines, functions, loops, and a primitive For loop.

IAL (International Algebraic Logic) started as the project later renamed ALGOL 58. The theoretical definition of the language is published. No compiler.

LISP (LISt Processing) was created n 1958 and released in 1960 by John McCarthy of MIT. LISP is the second oldest programming language still in common use. LISP was intended for writing artificial intelligence programs.

> “Interest in artificial intelligence first surfaced in the mid 1950. Linguistics, psychology, and mathematics were only some areas of application for AI. Linguists were concerned with natural language processing, while psychologists were interested in modeling human information and retrieval. Mathematicians were more interested in automating the theorem proving process. The common need among all of these applications was a method to allow computers to process symbolic data in lists.
>
> “IBM was one of the first companies interested in AI in the 1950s. At the same time, the FORTRAN project was still going on. Because of the high cost associated with producing the first FORTRAN compiler, they decided to include the list processing functionality into FORTRAN. The FORTRAN List Processing Language (FLPL) was designed and implemented as an extention

to FORTRAN.

"In 1958 John McCarthy took a summer position at the IBM Information Research Department. He was hired to create a set of requirements for doing symbolic computation. The first attempt at this was differentiation of algebraic expressions. This initial experiment produced a list of of language requirements, most notably was recursion and conditional expressions. At the time, not even FORTRAN (the only high-level language in existance) had these functions.

"It was at the 1956 Dartmouth Summer Research Project on Artificial Intelligence that John McCarthy first developed the basics behind Lisp. His motivation was to develop a list processing language for Artificial Intelligence. By 1965 the primary dialect of Lisp was created (version 1.5). By 1970 special-purpose computers known as Lisp Machines, were designed to run Lisp programs. 1980 was the year that object-oriented concepts were integrated into the language. By 1986, the X3J13 group formed to produce a draft for ANSI Common Lisp standard. Finally in 1992, X3J13 group published the American National Standard for Common Lisp." —Neal Ziring, The Language Guide, University of Michigan

"Some of the more significant features of the language are listed below:" —Neal Ziring, The Language Guide, University of Michigan

- **Atoms & Lists** - Lisp uses two different types of data structures, atoms and lists.
- **Atoms** are similar to identifiers, but can also be numeric constants
- **Lists** can be lists of atoms, lists, or any combination of the two
- **Functional Programming Style** - all computation is performed by applying functions to arguments. Variable declarations are rarely used.
- **Uniform Representation of Data and Code** - example: the list (A B C D)
  - a list of four elements (interpreted as data)
  - is the application of the function named A to the three parameters B, C, and D (interpreted as code)
- **Reliance on Recursion** - a strong reliance on recursion has allowed Lisp to be successful in many areas, including Artificial Intelligence.
- **Garbage Collection** - Lisp has built-in garbage collection, so programmers do not need to explicitly free dynamically allocated memory.

"55. A LISP programmer knows the value of everything, but the cost of nothing." —Alan Perlis, *Epigrams on Programming,* ACM's SIGPLAN Notices Volume 17, No. 9, September 1982, pages 7-13

Programming Languages: FORTRAN II; IAL; LISP
Operating Systems: UMES
Computers: UNIVAC II; IBM AN/FSQ-7 (400 kOPS)
Games: Tennis For Two (developed by William Higinnotham using an osciliscope and an analog computer)
Technology: integrated circuit

1959

COBOL (COmmon Business Oriented Language) was created in May 1959 by the Short Range Committee of the U.S. Department of Defense (DoD). The CODASYL committee (COnference on DAta SYstems Languages) worked from May 1959 to April 1960. Official ANSI standards included COBOL-68 (1968), COBOL-74 (1974), COBOL-85 (1985), and COBOL-2002 (2002). COBOL 97 (1997) introduced an object oriented version of COBOL. COBOL programs are divided into four divisions: identification, environment, data, and procedure. The divisions are further divided into sections. Introduced the RECORD data structure. Emphasized a verbose style intended to make it easy for business managers to read programs. Admiral Grace Hopper is recognized as the major contributor to the original COBOl language and as the inventor of compilers.

LISP 1.5 released in 1959.

" '**DYNAMO** is a computer program for translating mathematical models from an easy-to-understand notation into tabulated and plotted results. … A model written in DYNAMO consists of a number of algebraic relationships that relate the variables one to another.' Although similar to FORTRAN, it is easier to learn and understand. DYNAMO stands for DYNAmic MOdels. It was written by Dr. Phyllis Fox and Alexander L. Pugh, III, and was completed in 1959. It grew out of an earlier language called SIMPLE (for Simulation of Industrial Management Problems with Lots of Equations), written in 1958 by Richard K. Bennett." —Language Finger, Maureen and Mike Mansfield Library, University of Montana.

ERMA (Electronic Recording Method of Accounting), a magnetic ink and computer readable font, was created for the Bank of America.

Programming Languages: COBOL; DYNAMO; ERMA; LISP 1.5
Operating Systems: SHARE

Computers: IBM 1401

early 1960s

The early 1960s saw the introduction of time sharing and multi-processing.

Some operating systems from the early 1960s include: Admiral, B1, B2, B3, B4, Basic Executive System, BOS/360, Compatible Timesharing System (CTSS), EXEC I, EXEC II, Honeywell Executive System, IBM 1410/1710 OS, IBSYS, Input Output Control System, Master Control Program, and SABRE.

The first major transaction processing system was SABRE (Semi-Automatic Business Related Environment), developed by IBM and American Airlines.

multiprogramming

There is a huge difference in speed between I/O and running programs. In a single stream system, the processor remains idle for much of the time as it waits for the I/O device to be ready to send or receive the next piece of data.

The obvious solution was to load up multiple programs and their data and switch back and forth between programs or jobs.

When one job idled to wait for input or output, the operating system could automatically switch to another job that was ready.

system calls

The first operating system to introduce system calls was University of Machester's Atlas I Supervisor.

time sharing

The operating system could have additional reasons to rotate through jobs, including giving higher or lower priority to various jobs (and therefore a larger or smaller share of time and other resources). The Compatible Timesharing System (CTSS), first dmonstrated in 1961, was one of the first attempts at timesharing.

While most of the CTSS operating system was written in assembly language (all previous OSes were written in assembly for efficiency), the scheduler was written in the programming language MAD in order to allow safe and reliable experimentation with different scheduling algorithms. About half of the command programs for CTSS were also written in MAD.

Timesharing is a more advanced version of multiprogramming that gives many users the illusion that they each have complete control of the computer to themselves. The scheduler stops running programs based on a slice of time, moves on to the next program, and eventually returns back to the beginning of the list of programs. In little increments, each program gets their work done in a manner that appears to be simultaneous to the end users.

mid 1960s

Some operating systems from the mid-1960s include: Atlas I Supervisor, DOS/360, Input Output Selector, Master Control Program, and Multics.

The Atlas I Supervisor introduced spooling, interrupts, and virtual memory paging (16 pages) in 1962. Segmentation was introduced on the Burroughs B5000. MIT's Multics combined paging and segmentation.

The Compatible Timesharing System (CTSS) introduced email.

late 1960s

Some operating systems from the late-1960s include: BPS/360, CAL, CHIPPEWA, EXEC 3, EXEC 4, EXEC 8, GECOS III, George 1, George 2, George 3, George 4, IDASYS, MASTER, Master Control Program, OS/MFT, OS/MFT-II, OS/MVT, OS/PCP, and RCA DOS.

1960

ALGOL (ALGOrithmic Language) was released in 1960. Major releases in 1960 (ALGOL 60) and 1968 (ALGOL 68). ALGOL is the first block-structured labguage and is considered to be the first second generation computer language. This was the first programming language that was designed to be machine independent. ALGOL introduced such concepts as: block structure of code (marked by BEGIN and END), scope of variables (local variables inside blocks), BNF (Backus Naur Form) notation for defining syntax, dynamic arrays, reserved words, IF THEN ELSE, FOR, WHILE loop, the := symbol for assignment, SWITCH with GOTOs, and user defined data types. ALGOL became the most popular programming language in Europe in the mid- and late-1960s.

C.A.R. Hoare invents the **Quicksort**in 1960.

Programming Languages: ALGOL
Operating Systems: IBSYS
Computers: DEC PDP-1; CDC 1604; UNIVAC LARC (250 kFLOPS)

1961

The Compatible Timesharing System (CTSS), first dmonstrated in 1961, was one of the first attempts at timesharing.

While most of the CTSS operating system was written in assembly language (all previous OSes were written in assembly for efficiency), the scheduler was written in the programming language MAD in order to allow safe and reliable experimentation with different scheduling algorithms. About half of the command programs for CTSS were also written in MAD.

Operating Systems: CTSS, Burroughs MCP
Computers: IBM 7030 Stretch (1.2 MFLOPS)

1962

APL (A Programming Language) was published in the 1962 book *A Programming Language* by Kenneth E. Iverson and a subset was first released in 1964. The language APL was based on a notation that Iverson invented at Harvard University in 1957. APL was intended for mathematical work and used its own special character set. Particularly good at matrix manipulation. In 1957 it introduced the array. APL used a special character set and required special keyboards, displays, and printers (or printer heads).

FORTRAN IV is released in 1962.

Simula was created by Ole-Johan Dahl and Kristen Nygaard of the Norwegian Computing Center between 1962 and 1965. A compiler became available in 1964. Simula I and Simula 67 (1967) were the first object-oriented programming languages.

SNOBOL (StroNg Oriented symBOli Language) was created in 1962 by D.J. Farber, R.E. Griswold, and F.P. Polensky at Bell Telephone Laboratories. Intended for processing strings, the language was the first to use associative arrays, indexed by any type of key. Had features for pattern-matching, concatenation, and alternation. Allowed running code stored in strings. Data types: integer, real, array, table, pattern, and user defined types.

SpaceWarI, the first interactive computer game, was created by MIT students Slug Russel, Shag Graetz, and Alan Kotok on DEC's PDP-1.

The first operating system to introduce system calls was University of Machester's Atlas I Supervisor.

The Atlas I Supervisor introduced spooling, interrupts, and virtual memory paging (16 pages) in 1962. Segmentation was introduced on the Burroughs B5000. MIT's Multics combined paging and segmentation.

Operating Systems: GECOS
Programming Languages: APL; FORTRAN IV; Simula; SNOBOL
Computers: ATLAS, UNIVAC 1100/2200 (introduced two floating point formats, single precision and double precision; single precision: 36 bits, 1-bit sign, 8-bit exponent, and 27-bit significand; double precision: 36 bits, 1-bit sign, 11-bit exponent, and 60-bit significand), IBM 7094 (followed the UNIVAC, also had single and double precision numbers)
Games: Spacewar! (created by group of M.I.T. students on the DEC PDP-1)
Technology: RAND Corporation proposes the internet

1963

Work on PL/I starts in 1963.

> "**Data-Text** was the "original and most general problem-oriented computer language for social scientists." It has the ability to handle very complicated data processing problems and extremely intricate statistical analyses. It arose when FORTRAN proved inadequate for such uses. Designed by Couch and others, it was first used in 1963/64, then extensively revised in 1971. The Data-Text System was originally programmed in FAP, later in FORTRAN, and finally its own language was developed." —Language Finger, Maureen and Mike Mansfield Library, University of Montana.

Sketchpad, an interactive real time computer drawing system, was created in 1963 by Ivan Sutherland as his doctoral thesis at MIT. The system used a light pen to draw and manipulate geometric figures on a computer screen.

ASCII (American Standard Code for Information Interchange) was introduced in 1963.

Programming Languages: Data-Text
Computers: DEC PDP-6
Software: Sketchpad
Technology: mouse

1964

BASIC (Beginner's All-purpose Symbolic Instruction Code) was designed as a teaching language in 1963 by John George Kemeny and Thomas Eugene Kurtz of Dartmouth College. BASIC was intended to make it easy to learn programming. The first BASIC program was run at 4 a.m. May 1, 1964.

PL/I (Programming Language One) was created in 1964 at IBM's Hursley Laboratories in the United Kingdom. PL/I was intended to combine the scientific abilities of FORTRAN with the business capabilities of COBOL, plus additional facilities for systems programming. Also borrows from ALGOL 60. Originally called NPL, or New Programming Language. Introduces storage classes (automatic, static, controlled, and based), exception processing (On conditions), Select When Otherwise conditional structure, and several variations of the DO loop. Numerous data types, including control over precision.

RPG (Report Program Generator) was created in 1964 by IBM. Intended for creating commercial and business reports.

APL\360 implemented in 1964.

Operating Systems: DTSS, TOPS-10
Programming Languages: APL\360; BASIC; PL/I; RPG
Computers: IBM 360; DEC PDP-8; CDC 6600 (first supercomputer, scalar processor, 3 MFLOPS)
Technology: super computing

1965

Multics The Massachusetts Institute of Technology (MIT), AT&T Bell Labs, and General Electric attempted to create an experimental operating system called Multics for the GE-645 mainframe computer. AT&T intended to offer subscription-based computing services over the phone lines, an idea similar to the modern cloud approach to computing.

While the Multics project had many innovations that went on to become standard approaches for operating systems, the project was too complex for the time.

SNOBOL 3 was released in 1965.

Attribute grammars were created in 1965 by Donald Knuth.

Operating Systems: OS/360; Multics
Programming Languages: SNOBOL 3
Technology: time-sharing; fuzzy logic; packet switching; bulletin board system (BBS); email

1966

ALGOL W was created in 1966 by Niklaus Wirth. ALGOL W included RECORDs, dynamic data structures, CASE, passing parameters by value, and precedence of operators.

Euler was created in 1966 by Niklaus Wirth.

FORTRAN 66 was released in 1966. The language was rarely used.

ISWIM (If You See What I Mean) was described in 1966 in Peter J. Landin's article *The Next 700 Programming Languages* in the Communications of the ACM. ISWIM, the first purely functional language, influenced functional programming languages. The first language to use lazy evaluation.

LISP 2 was released in 1966.

Programming Languages: ALGOL W; Euler; FORTRAN 66; ISWIM
Computers: BESM-6

1967

Logo was created in 1967 (work started in 1966) by Seymour Papert. Intended as a programming language for children. Started as a drawing program. Based on moving a "turtle" on the computer screen.

Simula 67 was created by Ole-Johan Dahl and Kristen Nygaard of the Norwegian Computing Center in 1967. Introduced classes, methods, inheriteance, and objects that are instances of classes.

SNOBOL 4 (StroNg Oriented symBOli Language) was released in 1967.

CPL (Combined Programming Language) was created in 1967 at Cambridge and London Universities. Combined ALGOL 60 and functional language. Used polymorphic testing structures. Included the ANY type, lists, and arrays.

Operating Systems: ITS; CP/CMS; WAITS
Programming Languages: CPL; Logo; Simula 67; SNOBOL 4

1968

In 1968 a group of scientists and engineers from Mitre Corporation (Bedford, Massachusetts) created Viatron Computer company and an intelligent data terminal using an 8-bit LSI microprocessor from PMOS technology.

ALGOL 68 in 1968 introduced the =+ token to combine assignment and add, UNION, and CASTing of types. It included the IF THEN ELIF FI structure, CASE structure, and user-defined operators.

Forth was created by Charles H. Moore in 1968. Stack based language. The name "Forth" was a reference to Moore's claim that he had created a fourth generation programming language.

ALTRAN, a variant of FORTRAN, was released.

ANSI version of COBOL defined.

Edsger Dijkstra wrote a letter to the Communications of the ACM claiming that the use of GOTO was harmful.

Programming Languages: ALTRAN; ALGOL 68; Forth
Computers: DEC PDP-10
Technology: microprocessor; interactive computing (including mouse, windows, hypertext, and fullscreen word processing)

1969

In 1969 Viatron created the 2140, the first 4-bit LSI microprocessor. At the time MOS was used only for a small number of calculators and there simply wasn't enough worldwide manufacturing capacity to build these computers in quantity.

UNIX was created at AT&T Bell Telephone Laboratories (now an independent corporation known as Lucent Technologies) by a group that included Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, Michael Lesk and Joe Ossanna.

This group of researchers, the last of the AT&T employees involved in Multics, decided to attempt the goals of Multics on a much more simple scale.

Unix was originally called UNICS, for Uniplexed Information and Computing Service, a play on words variation of Multics, Multiplexed Information and Computing Service (*uni-* means "one", *multi-* means "many").

"What we wanted to preserve was not just a good environment in which to do programming, but a system around which a fellowship could form. We knew from experience that the essence of communal computing, as supplied by remote-access, time-shared machines, is not just to type programs into a terminal instead of a keypunch, but to encourage close communication," according to Dennis Ritchie.

UNIX was originally intended as a programmer's workbench. The original version was a single-user system. As UNIX spread through the academic community, more general purpose tools were added, turning UNIX into a general purpose operating system.

While Ken Thompson still had access to the Multics environment, he wrote simulations on Multics for UNIX's file and paging system.

Ken Thompson and Dennis Ritchie led a team of Bell Labs researchers (the team included Rudd Canaday) working on the PDP-7 to develop a hierarchial file system, computer processes, device files, a command-line interpreter, and a few small utility programs.

At the time, AT&T was prohibited from selling computers or software, but was allowed to develop its own software and computers for internal use. AT&T's consent decree with the U.S. Justice Department on monopoly charges was interpretted as allowing AT&T to release UNIX as an open source operating system for academic use. Ken Thompson, one of the originators of UNIX, took UNIX to the University of California, Berkeley, where students quickly started making improvements and modifications, leading to the world famous Berkeley Standard Distribution (BSD) form of UNIX.

UNIX quickly spread throughout the academic world, as it solved the problem of keeping track of many (sometimes dozens) of proprietary operating systems on university computers. With UNIX all of the computers from many different manufacturers could run the same operating system and share the same programs (recompiled on each processor).

BCPL (Basic CPL) was created in 1969 in England. Intended as a simplified version of CPL, includes the control structures For, Loop, If Then, While, Until Repeat, Repeat While, and Switch Case.

> "BCPL was an early computer language. It provided for comments between slashes. The name is condensed from "Basic CPL"; CPL was jointly designed by the universities of Cambridge and London. Officially, the "C" stood first for "Cambridge," then later for "Combined." -- Unofficially it was generally accepted as standing for Christopher Strachey, who was the main impetus behind the language." —Language Finger, Maureen and Mike Mansfield Library, University of Montana.

B (derived from BCPL) developed in 1969 by Ken Thompson of Bell Telephone Laboratories for use in systems programming for UNIX. This was the parent language of C.

SmallTalk was created in 1969 at Xerox PARC by a team led by Alan Kay, Adele Goldberg, Ted Kaehler, and Scott Wallace. Fully object oriented programming language that introduces a graphic environment with windows and a mouse.

RS-232-C standard for serial communication introduced in 1969.

Space Travel Ken Thompson ported the Space Travel computer game from Multics to a Digital Equipment Corporation (DEC) PDP-7 he found at Bell Labs.

ARPA creates **ARPAnet**, the forerunner of the Internet (originally hosted by UCLA, UC Santa Barbara, University of Utah, and Stanford Research Institute).

Operating Systems: ACP; TENEX/TOPS-20; work started on Unix
Programming Languages: B, BCPL; SmallTalk
Computers: CDC 7600 (36 MFLOPS)
Games: Space Travel (written by Jeremy Ben for Multics; when AT&T pulled out of the Multics project, J. Ben ported the program to FORTRAN running on GECOS on the GE 635; then ported by J. Ben and Dennis Ritchie in PDP-7 assembly language; the process of porting the game to the PDP-7 computer was the beginning of Unix)
Technology: ARPANET (military/academic precursor to the Internet); RS-232; networking; laser printer (invented by Gary Starkweather at Xerox)

early 1970s

Some operating systems from the early-1970s include: BKY, Chios, DOS/VS, Master Control Program, OS/VS1, and UNIX.

mid 1970s

Some operating systems from the mid-1970s include: CP/M, Master Control Program.

late 1970s

Some operating systems from the late-1970s include: EMAS 2900, General Comprehensive OS, VMS (later renamed OpenVMS), OS/MVS.

1970

Prolog (PROgramming LOGic) was created in 1972 in France by Alan Colmerauer with Philippe Roussel. Introduces Logic Programming.

Pascal (named for French religious fanatic and mathematician Blaise Pascal) was created in 1970 by Niklaus Wirth on a CDC 6000-series computer. Work started in 1968. Pascl was intended as a teaching language to replace BASIC, but quickly developed into a general purpose programming language. Programs compiled to a platform-independent intermediate P-code. The compiler for pascal was written in Pascal, an influential first in language design.

Forth used to write the program to control the Kitt Peaks telescope.

BLISS was a systems programming language developed by W.A. Wulf, D.B. Russell, and A.N. Habermann at Carnegie Mellon University in 1970. BLISS was a very popular systems programming language until the rise of C. The original compiler was noted for its optimizing of code. Most of the utilities for DEC's VMS operating system were written in BLISS-32. BLISS was a typeless language based on expressions rather than statements. Expressions produced values, and possibly caused other actions, such as modification of storage, transfer of control, or looping. BLISS had powerful macro facilities, conditional execution of statements, subroutines, built-in string functions, arrays, and some automatic data conversions. BLISS lacked I/O instructions on the assumption that systems I/O would actually be built in the language.

UNIX In 1970 Dennis Ritchie and Ken Thompson traded the promise to add text processing capabilities to UNIX for the use of a Digital Equipment Corporation (DEC) PDP-11/20. The initial version of UNIX, a text editor, and a text formatting program called `roff` were all written in PDP-11/20 assembly language.

ROFF Soon afterwards `roff` evolved into `troff` with full typesetting capability. The *UNIX Programmer's Manual* was published on November 3, 1971. The first commercial UNIX system was installed in early 1962 at the New York Telephone Co. Systems Development Center under the direction of Dan Gielan. Neil Groundwater build an Operational Support System in PDP-11/20 assembly language.

Operating Systems: Unix; RT-11; RSTS-11
Programming Languages: BLISS; Pascal; Prolog
Computers: Datapoint 2200; DEC PDP-11
Software: ROFF
Technology: dynamic RAM; flight data processor

1971

Manufacturers saw the benefit of MOS, starting with Intel's 1971 release of the 4-bit 4004 as the first commercially available microprocessor.

UNIX shell This first UNIX command shell, called the Thompson shell and abbreviated `sh`, was written by Ken Thompson at AT&T's Bell Labs, was much more simple than the famous UNIX shells that came along later. The Thompson shell was distributed with Versions 1 through 6 of UNIX, from 1971 to 1975.

Computers: Intel 4004 (four-bit microprocessor)
Software: UNIX shell
Games: Computer Space (first commercial vidoe game)
Technology: floppy disk; first electronic calculator (T1)

1972

In 1972 Rockwell released the PPS-4 microprocessor, Fairchild released the PPS-25 microprocessor, and Intel released the 8-bit 8008 microprocessor. All used PMOS.

C was developed from 1969-1972 by Dennis Ritchie of Bell Telephone Laboratories for use in systems programming for UNIX.

UNIX In 1972 work started on converting UNIX to C. The UNIX kernel was originally written in assembly language, but by 1973 it had been almost completely converted to the C language. At the time it was common belief that operating systems must be written in assembly in order to perform at reasonable speeds. This made Unix the world's first portable operating system, capable of being easily ported (moved) to any hardware. This was a major advantage for Unix and led to its widespread use in the multi-platform environments of colleges and universities. Writing UNIX in C allowed for easy portability to new hardware, which in turn led to the UNIX operating system being used on a wide variety of computers.

Pong, the first arcade video game, was introduced by Nolan Bushnell in 1972. His company was called Atari.

Operating Systems: VM/CMS; UNIX rewritten in C
Programming Languages: C
Computers: Intel 8008 (microprocessor); Rockwell PPS-4 (microprocessor); Fairchild PPS-25 (microprocessor)
Games: Pong; Magnavox Odysssey (first home video game console)
Technology: game console (Magnavox Odyssey); first scientific calculator (HP); first 32-bit minicomputer; first arcade video game

1973

In 1973 National released the IMP microprocessor using PMOS.

In 1973 Intel released the faster NMOS 8080 8-bit microprocessor, the first in a long series of microprocessors that led to the current Pentium.

ML (Meta Language) was created in 1973 by R. Milner of the University of Edinburgh. Functional language implemented in LISP.

Actor is a mathematical model for concurrent computation first published bby Hewitt in 1973.

> "Actor is an object-oriented programming language. It was developed by the Whitewater Group in Evanston, Ill." —Language Finger, Maureen and Mike Mansfield Library, University of Montana.

ARPA creates Transmission Control Protocol/Internet Protocol (TCP/IP) to network together computers for ARPAnet.

Programming Languages: ML
Computers: National IMP (microprocessor)
Software: Actor
Technology: TCP/IP; ethernet

1974

In 1974 Motorola released the 6800, which included two accumulators, index registers, and memory-mapped I/O. Monolithic Memories introduced bit-slice microprocessing.

SQL (Standard Query Language) was designed by Donald D. Chamberlin and Raymond F. Boyce of IBM in 1974.

AWK (first letters of the three inventors) was designed by Aho, Weinberger, and Kerninghan in 1974. Word processing language based on regular expressions.

Alphard (named for the brightest star in Hydra) was designed by William Wulf, Mary Shaw, and Ralph London of Carnegie-Mellon University in 1974. A Pascal-like language intended for data abstraction and verification. Make use of the "form", which combined a specification and an implementation, to give the programmer control over the impolementation of abstract data types.

> "Alphard is a computer language designed to support the abstraction and verification techniques required by modern programming methodology. Alphard's constructs allow a programmer to isolate an abstraction, specifying its behavior publicly while localizing knowledge about its implementation. It originated from studies at both Carnegie-Mellon University and the Information Sciences Institute." —Language Finger, Maureen and Mike Mansfield Library, University of Montana.

> "CLU began to be developed in 1974; a second version was designed in 1977. It consists of a group of modules. One of the primary goals in its development was to provide clusters which permit user-defined types to be treated similarly to built-in types." —Language Finger, Maureen and Mike Mansfield Library, University of Montana.

Operating Systems: MVS
Programming Languages: Alphard; AWK; CLU; SQL
Computers: Intel 8080 (microprocessor); Motorola 6800 (microprocessor); CDC STAR-100 (100 MFLOPS)
Technology: Telenet (first commercial version of ARPANET)

1975

In 1975 Texas Instruments introduced a 4-bit slice microprocessor and Fairchild introduced the F-8 microprocessor.

Scheme, based on LISP, was created by Guy Lewis Steele Jr. and Gerald Jay Sussman at MIT in 1975.

Tiny BASIC created by Dr. Wong in 1975 runs on Intel 8080 and Zilog Z80 computers.

RATFOR (RATional FORtran) created by Brian Kernigan in 1975. Used as a precompiler for FORTRAN. RATFOR allows C-like control structures in FORTRAN.

TENEX shell (also known as the TOPS shell) was created by Ken Greer by September 1975 and merged into the C shell in December 1983. Ken Greer based the shell on the TENEX operating system (hence, the "t" in `tsch`).

Programming Languages: RATFOR; Scheme; TENEX shell; Tiny BASIC
Computers: Altair 880 (first personal computer); Fairchild F-8 (microprocessor); MOS Technology 6502 (microprocessor); Burroughs ILLIAC IV (150 MFLOPS)
Technology: single board computer; laser printer (commercial release by IBM)

1976

Design System Language, a forerunner of PostScript, is created in 1976. The Forth-like language handles three dimensional databases.

SASL (Saint Andrews Static Language) is created by D. Turner in 1976. Intended for teaching functional programming. Based on ISWIM. Unlimited data structures.

CP/M, an operating system for microcomputers, was created by Gary Kildall in 1976.

PWB shell The PWB shell or Mashey shell, abbreviated `sh`, was a variation of the Thompson shell that had been augmented by John Mashey and others at Bell Labs. The Mashey shell was distributed with the Programmer's Workbench UNIX in 1976.

Operating Systems: CP/M
Programming Languages: Design System language; SASIL
Computers: Zilog Z-80 (microprocessor); Cray 1 (250 MFLOPS); Apple I
Software: PWB shell
Technology: inkjet printer; Alan Kay's Xerox NoteTaker developed at Xerox PARC

1977

Icon, based on SNOBOL, was created in 1977 by faculty, staff, and students at the University of Arizona under the direction of Ralph E. Griswold. Icon uses some programming structures similar to pascal and C. Structured types include list, set, and table (dictionary).

OPS5 was created by Charles Forgy in 1977.

FP was presented by John Backus in his 1977 Turing Award lecture *Can Programming be Liberated From the von Neumann Style? A Functional Style and its Algebra of Programs*.

Modula (MODUlar LAnguage) was created by Niklaus Wirth, who started work in 1977. Modula-2 was released in 1979.

Bourne shell The Bourne shell, created by Stephen Bourne at AT&T's Bell Labs as a scripting language, was released in 1977 as the default shell in the Version 7 Unix release distributed to colleges and universities. It gained popularity with the publication of *The UNIX Programming Environment* by Brian W. Kernighan and Rob Pike. The book was the first commercially published tutorial on shell programming.

BSD UNIX The University of California, Berkeley, released the Berkeley Software Distribution (BSD) version of UNIX, based on the 6th edition of AT&T UNIX.

Programming Languages: Bourne shell; FP; Icon; Modula; OPS5
Computers: DEC VAX-11; Apple II; TRS-80; Commodore PET; Cray 1A
Games: Atari 2600 (first popular home video game consle)

1978

CSP was created in 1978 by C.A.R. Hoare.

> "C.A.R. Hoare wrote a paper in 1978 about parallel computing in which he included a fragment of a language. Later, this fragment came to be known as CSP. In it, process specifications lead to process creation and coordination. The name stands for Communicating Sequential Processes. Later, the separate computer language Occam was based on CSP." —Language Finger, Maureen and Mike Mansfield Library, University of Montana.

C Shell The C shell, abbreviated `csh`, was created by Bill Joy, a graduate student at the University of California, Berkeley. With additional work by Michael Ubell, Eric Allman, Mike O'Brien, and Jim Kulp, it was released in the 2BSD release of BSD Unix in 1978.

It became common to use the C shell for everyday interactive computing and to use the Bourne shell for script programming.

Operating Systems: Apple DOS 3.1; VMS (later renamed OpenVMS)

Programming Languages: C shell; CSP
Computers: Intel 8086 (microprocessor)
Games: Space Invaders (arcade game using raster graphics; so popular in Japan that the government has to quadruple its supply of Yen)
Technology: LaserDisc

1979

Modula-2 was released in 1979. Created by Niklaus Wirth, who started work in 1977.

VisiCalc (VISIble CALculator) was created for the Apple II personal computer in 1979 by Harvard MBA candidate Daniel Bricklin and programmer Robert Frankston.

REXX (REstructured eXtended eXecutor) was designed by Michael Cowlishaw of IBM UK Laboratories. REXX was both an interpretted procedural language and a macro language. As a maco language, REXX can be used in application software.

Work started on **C with Classes**, the language that eventually became C++.

Programming Languages: C with Classes; Modula-2; REXX
Computers: Motorola MC68000 (microprocessor); Intel 8088 (microprocessor)
Software: VisiCalc
Games: Lunar Lander (arcade video game, first to use vector graphics); Asteroids (vector arcade game); Galaxian (raster arcade game, color screen); first Multi-USer Dungeon (MUD, written by Roy Trubshaw , a student at Essex University, forrunner of modern massively multiplayer games); Warrior (first head-to-head arcade fighting game)
Technology: first spreadsheet (VisiCalc); object oriented programming; compact disk; Usenet discussion groups

1980s

Some operating systems from the 1980s include: AmigaOS, DOS/VSE, HP-UX, Macintosh, MS-DOS, and ULTRIX.

The 1980s saw the commercial release of the graphic user interface, most famously the Apple Macintosh, Commodore Amiga, and Atari ST, followed by MIT's X Window System (X11) and Microsoft's Windows.

1980

dBASE II was created in 1980 by Wayne Ratliff at the Jet Propulsion Laboratories in Pasadena, California. The original version of the language was called **Vulcan**. Note that the first version of dBASE was called dBASE II.

SmallTalk-80 released.

Operating Systems: OS-9
Computers: Commodore VIC-20; ZX80; Apple III
Software: dBASE II
Games: Battlezone (vector arcade video game, dual joystick controller and periscope-like viewer); Berzerk (raster arcade video game, used primative speech synthesis); Centipede (raster arcade video game, used trackball controller); Missile Command (raster arcade video game, used trackball controller); Defender (raster arcade video game); Pac-Man (raster arcade video game); Phoenix (raster arcade video game, use of musical score); Rally-X (raster arcade video game, first game to have a bonus round); Star Castle (vector arcade video game, color provided by transparent plastic screen overlay); Tempest (vector arcade video game, first color vector game); Wizard of Wor (raster arcade video game)

1981

Relational Language was created in 1981 by Clark and Gregory.

Operating Systems: MS-DOS; Pilot
Programming Languages: Relational Language
Computers: 8010 Star; ZX81; IBM PC; Osborne 1 (first portable computer); Xerox Star; MIPS I (microprocessor); CDC Cyber 205 (400 MFLOPS)
Games: Donkey Kong (raster arcade video game); Frogger (raster arcade video game); Scramble (raster arcade video game, horizontal scrolling); Galaga (raster arcade video game); Ms. Pac-Man (raster arcade video game); Qix (raster arcade video game); Gorf (raster arcade video game, synthesized speech); Zork (first adventure game)
Technology: portable PC; ISA bus; CGA video card

1982

ANSI C The American National Standards Institute (ANSI) formed a technical subcommittee, X3J11, to create a standard for the C language and its run-time libraries.

InterPress, the forerunner of PostScript, was created in 1982 by John Warnock and Martin Newell at Xerox PARC.

Operating Systems: SunOS
Programming Languages: ANSI C; InterPress
Computers: Cray X-MP; BBC Micro; Commodore C64 (first home computer with a dedicated sound chip); Compaq Portable; ZX Spectrum; Atari 5200; Intel 80286 (microprocessor)
Games: BurgerTime (raster arcade video game); Dig Dug (raster arcade video game); Donkey Kong Junior (raster arcade video game); Joust (raster arcade video game); Moon Patrol (raster arcade video game, first game with parallax scrolling); Pole Position (raster arcade video game); Q*bert (raster arcade video game); Robotron 2084 (raster arcade video game, dual joystick); Time Pilot (raster arcade video game); Tron (raster arcade video game); Xevious (raster arcade video game, first game promoted with a TV commercial); Zaxxon (raster arcade video game, first game to use axonometric projection)
Technology: MIDI; RISC; IBM PC compatibles

1983

Ada was first released in 1983 (ADA 83), with major releases in 1995 (ADA 95) and 2005 (ADA 2005). Ada was created by the U.S. Department of Defense (DoD), originally intended for embedded systems and later intended for all military computing purposes. Ada is named for Augusta Ada King, the Countess of Lovelace, the first computer programmer inmodern times.

Concurrent Prolog was created in 1983 by Shapiro.

Parlog was created in 1983 by Clark and Gregory.

C++ was developed in 1983 by Bjarne Stroustrup at Bell Telephone Laboratories to extend C for object oriented programming.

Turbo Pascal, a popular Pascal compiler, was released.

The University of California at Berkeley released a version of UNIX that included TCP/IP.

tcsh The improved C shell, abbreviated `tcsh`, created by Ken Greer, was merged into the C shell in December 1983. Ken Greer based the shell on the TENEX operating system (hence, the “t” in `tsch`). Mike Ellis, Paul Placeway, and Christos Zoulas made major contributions to `tcsh`.

Korn shell The Korn shell, abbreviated `ksh`, was created by David Korn at AT&T’s Bell Labs and announced at USENIX on July 14, 1983. Mike Veach and Pat Sullivan were also early contributors. The Korn shell added C shell features to the Bourne shell.

Operating Systems: Lisa OS
Programming Languages: Ada; C++; Concurrent Prolog; Korn shell; Parlog; tcsh; Turbo Pascal
Computers: Apple IIe; Lisa; IBM XT; IBM PC Jr; ARM (microprocessor); Cray X-MP/4 (941 MFLOPS)
Games: Dragon’s Lair (raster arcade video game, first video game to use laserdisc video; note that the gambling device Quarterhorse used the laserdisc first); Elevator Action (raster arcade video game); Gyruss (raster arcade video game, used the musical score Toccata and Fugue in D minor by Bach); Mappy (raster arcade video game, side scrolling); Mario Bros. (raster arcade video game); Spy Hunter (raster arcade video game, musical score Peter Gunn); Star Wars (vector arcade video game, digitized samples from the movie of actor’s voices); Tapper (raster arcade video game); Lode Runner (Apple][E); Journey (arcade video game includes tape of the song *Separate Ways* leading to licensed music in video games)
Technology: math coprocessor; PC harddisk

1984

Objective C, an extension of C inspired by SmallTalk, was created in 1984 by Brad Cox. Used to write NextStep, the operating system of the Next computer.

Standard ML, based on ML, was created in 1984 by R. Milner of the University of Edinburgh.

PostScript was created in 1984 by John Warnock and Chuck Geschke at Adobe.

When the AT&T broke up in 1984 into “Baby Bells” (the regional companies operating local phone service) and the central company (which had the long distance business and Bell Labs), the U.S. government allowed them to start selling computers and computer software. UNIX broke into the System V (sys-five) and Berkeley Standard Distribution (BSD) versions. System V was the for pay version and BSD was the free open source version.

AT&T gave academia a specific deadline to stop using "encumbered code" (that is, any of AT&T's source code anywhere in their versions of UNIX). This led to the development of free open source projects such as FreeBSD, NetBSD, and OpenBSD, as well as commercial operating systems based on the BSD code.

Meanwhile, AT&T developed its own version of UNIX, called System V. Although AT&T eventually sold off UNIX, this also spawned a group of commercial operating systems known as Sys V UNIXes.

UNIX quickly swept through the commercial world, pushing aside almost all proprietary mainframe operating systems. Only IBM's MVS and DEC's OpenVMS survived the UNIX onslaught.

Some of the famous official UNIX versions include Solaris, HP-UX, Sequent, AIX, and Darwin. Darwin is the UNIX kernel for Apple's OS X, AppleTV, and iOS (used in the iPhone, iPad, and iPod).

The BSD variant started at the University of California, Berkeley, and includes FreeBSD, NetBSD, OpenBSD, and DragonFly BSD.

Most modern versions of UNIX combine ideas and elements from both Sys-V and BSD.

Other UNIX-like operating systems include MINIX and Linux.

In 1984, Jim Gettys of Project Athena and Bob Scheifler of the MIT Laboratory for Computer Science collaborated on a platform-independent graphcis system to link together heterogeneous multi-vendor systems. Project Athena was a joint project between Digital Equipment Corporation (DEC), IBM, and MIT. X version 1 was released in May 1984.

Operating Systems: GNU project started; MacOS 1, X Window System (X11)
Programming Languages: Objective C; PostScript; Standard ML
Computers: Apple Macintosh; IBM AT; Apple IIc; MIPS R2000 (microprocessor); M-13 (U.S.S.R., 2.4 GFLOPS); Cray XMP-22
Software: Apple MacWrite; Apple MacPaint
Games: 1942 (raster arcade video game); Paperboy (raster arcade video game, unusual controllers, high resolution display); Punch-Out (raster arcade video game, digitized voice, dual monitors)
Technology: WYSIWYG word processing; LaserJet printer; DNS (Domain Name Server); IDE interface

1985

Paradox was created in 1985 as a competitor to the dBASE family of relational data base languages.

PageMaker was created for the Apple Macintosh in 1985 by Aldus.

The Intel 80386 was the first Intel x86 microprocessor with a 32-bit instruction set and an MMU with paging.

Operating Systems: GEM; AmigaOS; AtariOS; WIndows 1.0; Mac OS 2
Programming Languages: Paradox
Computers: Cray-2/8 (3.9 GFLOPS); Atari ST; Commodore Amiga; Apple Macintosh XL; Intel 80386 (microprocessor); Sun SPARC (microprocessor)
Software: Apple Macintosh Office; Aldus PageMaker (Macintosh only)
Games: Super Mario Bros.; Tetris (puzzle game; invented by Russian mathematician Alexey Pajitnov to test equipment that was going to be used for artificial intelligence and speech recognition research); Excitebike (first game with battery backup for saving and loading game play)
Technology: desktop publishing; EGA video card; CD-ROM; expanded memory (for IBM-PC compatibles; Apple LaserWriter

1986

Eiffel (named for Gustave Eiffel, designer of the Eiffel Tower) was released in 1986 by Bertrand Meyer. Work started on September 14, 1985.

> "Eiffel is a computer language in the public domain. Its evolution is controlled by Nonprofit International Consortium for Eiffel (NICE), but it is open to any interested party. It is intended to treat software construction as a serious engineering enterprise, and therefore is named for the French architect, Gustave Eiffel. It aims to help specify, design, implement, and change quality software." —Language Finger, Maureen and Mike Mansfield Library, University of Montana.

GAP (Groups, Algorithms, and Programming) was developed in 1986 by Johannes Meier, Werner Nickel, Alice Niemeter, Martin Schönert, and others. Intended to program mathematical algorithms.

CLP(R) was developed in 1986.

In 1986, Maurice J. Bach of AT&T Bell Labs published *The Design of the UNIX Operating System*, which described the System V Release 2 kernel, as well as some new features from release 3 and BSD.

Operating Systems: Mach; AIX; GS-OS; HP-UX; Mac OS 3
Programming Languages: CLP(R); Eiffel; GAP
Computers: Apple IIGS; Apple Macintosh Plus; Amstrad 1512; ARM2 (microprocessor); Cray XMP-48
Software: Apple Macintosh Programmer's Workshop
Games: Metroid (one of first games to have password to save game proogress; first female protagonist in video games, non-linear game play; RPG)
Technology: SCSI

1987

CAML (Categorical Abstract Machine Language) was created by Suarez, Weiss, and Maury in 1987.

Perl (Practical Extracting and Report Language) was created by Larry Wall in 1987. Intended to replace the Unix shell, Sed, and Awk. Used in CGI scripts.

HyperCard was created by William Atkinson in 1987. **HyperTalk** was the scripting language built into HyperCard.

Thomas and John Knoll created the program Display, which eventually became PhotoShop. The program ran on the Apple Macintosh.

Adobe released the first version of Illustrator, running on the Apple Macintosh.

Minix Andrew S. Tanenbaum released MINIX, a simplified version of UNIX intended for academic instruction.

Operating Systems: IRIX; Minix; OS/2; Windows 2.0; MDOS (Myarc Disk Operating System); Mac OS 4; Mac OS 5
Programming Languages: CAML; HyperTalk; Perl
Computers: Apple Macintosh II; Apple Macintosh SE; Acorn Archimedes; Connection Machine (first massive parallel computer); IBM PS/2; Commodore Amiga 500; Nintendo Entertainment System
Software: Apple Hypercard; Apple MultiFinder; Adobe Illustrator (Macintosh only; later ported to NeXT, Silicon Graphics IRIX, Sun Solaris, and Microsoft Windows); Display (which eventually became ImagePro and then Photoshop; Macintosh only); QuarkXPress (Macintosh only)
Games: Final Fantasy (fantasy RPG); The Legend of Zelda (first free form adventure game); Gran Turismo (auto racing); Mike Tyson's Punch Out (boxing sports game)
Technology: massive parallel computing; VGA video card; sound card for IBM-PC compatibles; Apple Desktop Bus (ADB)

1988

CLOS, an object oriented version of LISP, was developed in 1988.

Mathematica was developed by Stephen Wolfram.

Oberon was created in 1986 by Niklaus Wirth.

Refined version of Design becomes ImagePro for the Apple Macintosh.

Operating Systems: OS/400; Mac OS 6
Programming Languages: CLOS; Mathematica; Oberon
Computers: Cray Y-MP; Apple IIc Plus
Software: ImagePro (which eventually became Photoshop; Macintosh only)
Games: Super Mario Bros. 2; Super Mario Bros 3 (Japanese release); Contra (side-scrolling shooter); Joh Madden Football (football sports game)
Technology: optical chip; EISA bus

1989

ANSI C The American National Standards Institute (ANSI) completed the official ANSI C, called the *American National Standard X3.159-1989*.

HTML was developed in 1989.

Miranda (named for a character by Shakespeare) was created in 1989 by D. Turner. Based on SASL and ML. Lazy evaluation and embedded pattern matching.

Standard Interchange Languagewas developed in 1989.

BASH, which stands for Bourne Again SHell, was created by Brian Fox for the Free Software Foundation and first released on June 7, 1989. `bash` combined features from the Bourne shell, the C shell, and the Korn shell. `bash` is now the primary shell in both Linux and Mac OS X.

Operating Systems: NeXtStep; RISC OS; SCO UNIX
Programming Languages: ANSI C; BASH; HTML; Miranda; Standard Interchage Language
Computers: Intel 80486 (microprocessor); ETA10-G/8 (10.3 GFLOPS)
Software: Microsoft Office; first (pre-Adobe) version of Photoshop (Macintosh only)
Games: Sim; SimCity; Warlords (four-player shooter)
Technology: ATA interface, World Wide Web

1990s

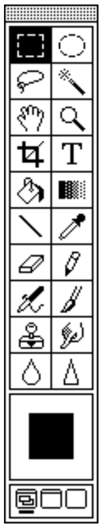
Some operating systems from the 1990s include: BeOS, BSDi, FreeBSD, NeXT, OS/2, Windows 95, Windows 98, and Windows NT.

1990

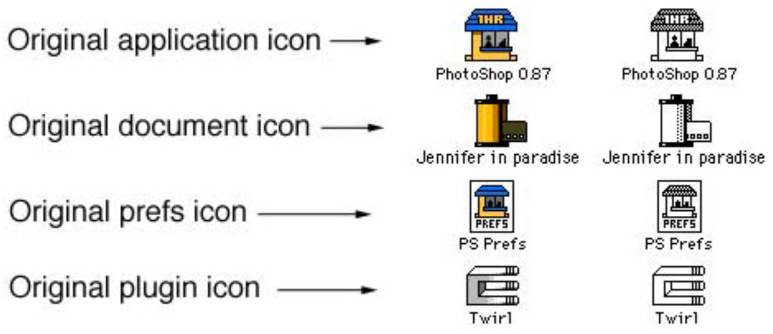
Haskell was developed in 1990.

Tim Berners-Lee of the European CERN laboratory created the World Wide Web on a NeXT computer.

Z shell The Z shell, abbreviated `zsh`, was written by Paul Flastad in 1990 when he was a student at Princton University.



In February of 1990, Adobe released the first version of the program PhotoShop (for the Apple Macintosh). Version 2.0 of Photoshop, code named Fast Eddy, ships in the fall of 1990. Version 1.07 toolbar to right and original icons designed by John Knoll below.



Operating Systems: BeOS; OSF/1
Programming Languages: Haskell; Z shell
Computers: NEC SX-3/44R (Japan, 23.2 GFLOPS); Cray XMS; Cray Y-MP 8/8-64 (first Cray supercomputer to use UNIX); Apple Macintosh Classic; Neo Geo; Super Nintendo Entertainment System
Software: Adobe Photoshop (Macintosh only)
Games: Final Fantasy III released in Japan (fantasy RPG); Super Mario Bros 3 (Japanese release); Wing Commander (space combat game)
Technology: SVGA video card; VESA driver

1991

Python (named for Monty Python Flying Circus) was created in 1991 by Guido van Rossum. A scripting language with dynamic types intended as a replacement for Perl.

Pov-Ray (Persistence of Vision) was created in 1991 by D.B.A. Collins and others. A language for describing 3D images.

Visual BASIC, a popular BASIC compiler, was released in 1991.

Linux operating system was first released on September 17, 1991, by Finnish student Linus Torvalds. With the permission of Andrew S. Tanenbaum, Linus Torvalds started work with MINIX. There is no MINIX source code left in Linux.

Linus Torvalds started work on open source Linux as a college student. After Mac OS X, Linux is the most widely used variation of UNIX.

Linux is technically just the kernel (innermost part) of the operating system. The outer layers consist of GNU tools. GNU was started to guarantee a free and open version of UNIX and all of the major tools required to run UNIX.

Operating Systems: Linux kernel; Mac OS 7
Programming Languages: Pov-Ray; Python; Visual BASIC
Computers: Apple PowerBook; PowerPC (microprocessor); PARAM 8000 (India, created by Dr. Vijay Bhatkar, 1GFLOP)
Games: Street Fighter II (shooter); The Legend of Zelda: A Link to the Past (fantasy RPG); Sonic the Hedgehog; Sid Meyer's Civilization
Technology: CD-i

1992

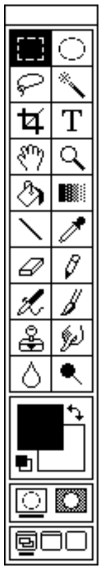
Dylan was created in 1992 by Apple Computer and others. Dylan was originally intended for use with the Apple Newton, but wasn't finished in time.

Oak, the forerunner of Java, was developed at Sun Microsystems.

In 1992 Unix System Laboratories sued Berkeley Software Design, Inc and the Regents of the University of California to try to stop the distribution of BSD UNIX. The case was settled out of court in 1993 after the judge expressed doubt over the validity of USL's intellectual property.

Operating Systems: Solaris; Windows 3.1; OS/2 2.0; SLS Linux; Tru64 UNIX
Programming Languages: Dylan; Oak
Computers: Cray C90 (1 GFLOP)
Games: Wolfenstein 3D (ffirst fully 3D rendered game engine); Mortal Kombat; NHLPA 93 (multiplayer hockey sports game); Dune II (first real-time strategy game)

1993



"Vendors such as Sun, IBM, DEC, SCO, and HP modified Unix to differentiate their products. This splintered Unix to a degree, though not quite as much as is usually perceived. Necessity being the mother of invention, programmers have created development tools that help them work around the differences between Unix flavors. As a result, there is a large body of software based on source code that will automatically configure itself to compile on most Unix platforms, including Intel-based Unix.

Regardless, Microsoft would leverage the perception that Unix is splintered beyond hope, and present Windows NT as a more consistent multi-platform alternative." —Nicholas Petreley, "The new Unix alters NT's orbit", NC World

AppleScript, a scripting language for the Macintosh operating system and its application softweare, was released by Apple Computers.

Version 2.5.1 of Photoshop released in 1993. It was one of the first programs to run native on a PowerPC chip. First release of Windows version of Photoshop in April 1993. Toolbar for Photoshop 2.5.1 to the right.

Operating Systems: Windows NT 3.1; Stackware Linux; Debian GNU/Linux; Newton
Programming Languages: AppleScript
Computers: Cray EL90; Cray T3D; Apple Newton; Apple Macintosh TV; Intel Pentium (microprocessor, 66MHz); Thinking Machines CM-5/1024 (59.7 GFLOPS); Fujitsu Numerical Wind Tunnel (Japan, 124.50 GFLOPS); Intel Paragon XP/S 140 (143.40 GFLOPS)
Games: Myst (first puzzle-based computer adventure game; CD-ROM game for Macintosh); Doom (made the first person shooter genre popular, pioneering work in immersive 3D graphics, networked multiplayer gaming, and support for customized additions and modifications; also proved that shareware distribution could work for game distribution); U.S. Senator Joseph Lieberman holds Congressional hearings and attempts to outlaw violent games
Technology: MP3

1994

Work continued on **Java** with a version designed for the internet.

Version 3.0 of Photoshop released in 1994. It included Layers.

Operating Systems: Red Hat Linux
Programming Languages: Java
Computers: Cray J90; Apple Power Macintosh; Sony PlayStation; Fujitsu Numerical Wind Tunnel (Japan, 170.40 GFLOPS)
Games: Super Metroid (RPG)
Technology: DNA computing

1995

Java (named for coffee) was created by James Gosling and others at Sun Microsystems and released for applets in 1995. Original work started in 1991 as an interactive language under the name Oak. Rewritten for the internet in 1994.

JavaScript (originally called LiveScript) was created by Brendan Elch at Netscape in 1995. A scripting language for web pages.

PHP (PHP Hypertext Processor) was created by Rasmus Lerdorf in 1995.

Ruby was created in 1995 by Yukihiro Matsumoto. Alternative to Perl and Python.

Delphi, a variation of Object Pacal, was released by Borland

Operating Systems: OpenBSD; OS/390; Windows 95
Programming Languages: Delphi; Java; JavaScript; PHP; Ruby
Computers: BeBox; Cray T3E; Cray T90; Intel Pentium Pro (microprocessor); Sun UltraSPARC (microprocessor)
Software: Microsoft Bob
Games: Chrono Trigger (fantasy RPG); Command & Conquer (real time strategy game)
Technology: DVD; wikis

1996

UML (Unified Modeling Language) was created by Grady Booch, Jim Rumbaugh, and Ivar Jacobson in 1996 by combining the three modeling languages of each of the authors.

Version 4.0 of Photoshop released in 1996. It included a controversial change in the key commands.

Operating Systems: MkLinux
Programming Languages: UML
Computers: Hitachi SR2201/1024 (Japan, 220.4 GFLOPS); Hitachi/Tsukuba CP=PACS/2048 (Japan, 368.2 GFLOPS)
Games: Tomb Raider and Lara Croft, Pokémon; Quake (first person shooter using new 3D rendering on daughter boards); Super Mario 64 (3D rendering)
Technology: USB

1997

REBOL (Relative Expression Based Object language) was created by Carl SassenRath in 1997. Extensible scripting language for internet and distributed computing. Has 45 types that use the same operators.

ECMAScript (named for the European standards group E.C.M.A.) was created in 1997.

Alloy, a structural modelling language, was developed at M.I.T.

Rhapsody The first developer version of Mac OS X released.on August 31, 1997.

To date, the most widely used desktop version of UNIX is Apple’s Mac OS X, combining the ground breaking object oriented NeXT with some of the user interface of the Macintosh.

Operating Systems: Mac OS 8; Rhapsody
Programming Languages: Alley; ECMAScript; REBOL

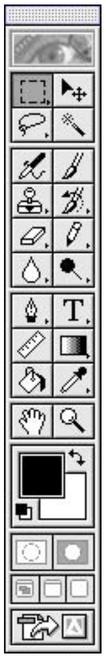
Computers: AMD K6 (microprocessor); Intel Pentium II (microprocessor); PalmPilot; Intel ASCI Red/9152 (1.338 TFLOPS)
Software: AOL Instant Messenger
Games: Final fantasy VII; Goldeneye 007 (one of the few successful movie to game transitions; based on 1995 James Bond movie *Goldeneye*); Castlevania: Symphony of the Night (2D fantasy RPG)
Technology: web blogging

1998

Version 5.0 of Photoshop released in 1998. It included the History palette.

Operating Systems: Windows 98
Computers: Apple iMac; Apple iMac G3; Intel Xeon (microprocessor)
Games: The Legend of Zelda: Ocarina of Time (fantasy RPG); Metal Gear Solid; Crash Bandicoot: Warped

1999



Mac OS X Mac OS X Server 1.0, intended as a server operating system and not for general desktop use, and Mac OS X Developer Preview were released on March 16, 1999.

Version 5.5 of Photoshop released in 1999. It was the first web ready version of Photoshop. Toolbar for Photoshop 5.5 to the right.

Operating Systems: Mac OS 9; Mac OS X Server
Computers: PowerMac; AMD Athlon (microprocessor); Intel Pentium III (microprocessor); BlackBerry; Apple iBook; TiVo; Intel ASCI Red/9632 (2.3796 TFLOPS)

2000s

Some operating systems from the 2000s include: Mac OS X, Syllable, Windows 2000, Windows Server 2003, Windows ME, and Windows XP.

2000

D, designed by Walter Bright, is an object-oriented language based on C++, Java, Eiffel, and Python.

C# was created by Anders Hajlsberg of Microsoft in 2000. The main language of Microsoft’s .NET.

RELAX (REgular LAnguage description for XML) was designed by Murata Makoto.

Mac OS X The Public Beta version of Mac OS X was released on September 13, 2000.

Operating Systems: Mac OS 9; Windows ME; Windows 2000
Programming Languages: C#; D; RELAX
Computers: Intel Pentium 4 (microprocessor, over 1 GHz); Sony PlayStation 2; IBM ASCI White (7.226 TFLOPS)
Games: Tony Hawk’s Pro Skater 2 (skateboarding sports game); Madden NFL 2001 (football sports game)
Technology: USB flash drive

2001

AspectJ (Aspect for Java) was created at the Palo Alto Research Center in 2001.

Scriptol (Scriptwriter Oriented Language) was created by Dennis G. Sureau in 2001. New control structuress include for in, while let, and scan by. Variables and literals are objects. Supports XML as data structure.

Mac OS X 10.0 Mac OS X 10.0 Cheetah was released on March 24, 2001.

Mac OS X 10.1 Mac OS X 10.1 Puma was released on September 25, 2001. It included DVD playback.

Operating Systems: Mac OS X 10.0 Cheetah; Mac OS X 10.1 Puma; Windows XP; z/OS
Programming Languages: AspectJ; Scriptol
Computers: Nintendo GameCube; Apple iPod; Intel Itanium (microprocessor); Xbox
Games: Halo
Technology: blade server

2002

Mac OS X 10.2 Mac OS X 10.2 Jaguar was released on August 24, 2002. It included the Quartz Extreme graphics engine, a system-wide repository for contact information in the Address Book, and an instant messaging client iChat.

Operating Systems: Mac OS X 10.2 Jaguar
Computers: Apple eMac; Apple iMac G4; Apple XServe; NEC Earth Simulator (Japan, 35.86 TFLOPS)

2003

Mac OS X 10.3 Mac OS X 10.3 Panther was released on October 24, 2003. It included a brushed metal look, fast user switching, Exposé (a Window manager), Filevault, Safari web browser, iChat AV (which added videoconferencing), imporved PDF rendering, and better Microsoft Windows interoperability.

Operating Systems: Windows Server 2003; Mac OS X 10.3 Panther
Computers: PowerPC G5; AMD Athlon 64 (microprocessor); Intel Pentium M (microprocessor)
Games: Final Fantasy Crystal Chronicles

2004

Scala was created February 2004 by Ecole Polytechnique Federale de Lausanne. Object oriented language that implements Python features in a Java syntax.

Programming Languages: Scala
Computers: Apple iPod Mini; Apple iMac G5; Sony PlayStation Portable; IBM Blue Gene/L (70.72 TFLOPS)
Games: Fable
Technology: DualDisc; PCI Express; USB FlashCard

2005

Job Submission Description Language.

Mac OS X 10.4 Mac OS X 10.4 Tiger was released on April 29, 2005. It included Spotlight (a find routine), Dashboard, Smart Folders, updated Mail program with Smart mailboxes, QuickTime 7, Safari 2, Automator, VoiceOver, Core Image, and Core Video.

Operating Systems: Mac OS X 10.4 Tiger
Programming Languages: Job Submission Description Language
Computers: IBM System z9; Apple iPod Nano; Apple Mac Mini; Intel Pentium D (microprocessor); Sun UltraSPARC IV (microprocessor); Xbox 360
Games: Lego Star Wars

2006

Computers: Apple Intel-based iMac; Intel Core 2 (microprocessor); Sony PlayStation 3
Technology: Blu-Ray Disc

2007

Mac OS X 10.5 Mac OS X 10.5 Leopard was released on October 26, 2007. It included an updated Finder, Time Machine, Spaces, Boot Camp, full support for 64-bit applications, new features in Mail and iChat, and new security features.

Operating Systems: Apple iOS (for iPhone); Mac OS X 10.5 Leopard; Windows Vista
Computers: AMD K10 (microprocessor), Apple TV; Apple iPhone; Apple iPod Touch; Amazon Kindle

2008

Operating Systems: Google Android
Computers: Android Dev Phone 1; BlackBerry Storm; Intel Atom (microprocessor); MacBook Air; IBM RoadRunner (1.026 PFLOPS); Dhruva (India, 6 TFLOPS)

2009

Mac OS X 10.6 Mac OS X 10.6 Snow Leopard was released on August 28, 2009. it included the SquirrelFish javaScript interpreter, a rewrite of Finder in the Cocoa API, faster Time Machine backups, more reliable disk ejects, a more powerful version on Preview, a faster version of Safari, Microsoft Exchange Server support for Mail, iCal, and Address Book, QuickTime X, Grand Central Dispatch for using multi-core processors, and OpenCL support.

Operating Systems: Windows 7; Mac OS X 10.6 Snow Leopard
Computers: Motorola Driod; Palm Pre; Cray XT5 Jaguar (1.759 PFLOPS)

2010

Computers: Apple iPad; IBM z196 (microprocessor); Apple iPhone 4; Kobo eReader; HTC Evo 4G

2011

Mac OS X 10.7 Mac OS X 10.7 Lion was released on July 20, 2011. it included Launchpad, auto-hiding scrollbars, and Mission Control.

Operating Systems: Mac OS X 10.7 Lion

2012

Mac OS X 10.8 Mac OS X 10.8 Mountain Lion was released on July 25, 2012. It included Game Center, Notification Center, iCloud support, and more Chinese features.

Operating Systems: Mac OS X 10.8 Mountain Lion, Windows 8

Forth-like routines for UNIX/Linux shell

appendix B

summary

This chapter looks at a set of Forth-like routines for use in the BASH UNIX (Linux, Mac OS X) shell.

basic approach

This supplementary material describes a method for creating Forth-like routines for use in the BASH shell.

The data and return stacks will be stored using a pair of shell variables, one of which is an array and the other is an ordinary scalar variable.

The Forth-like routines will be created as shell functions.

Everything will be stored in a shell function file that can be loaded manually or loaded as part of the normal shell start-up.

Eventually I intend to write two sets of C source code, one that people can commpile into their own copy of the official BASH source code and one that is part of an entirely new open source shell project that doesn't have the GNU infecting license. This alternative shell will be designed so that it can run on bare metal, allowing it to be used to deploy embedded systems.

stacks

Forth uses two stacks: the main data stack and the return stack.

For the purposes of the BASH shell, we will implement both stacks with a combination of a top of stack pointer variable and an array for the stack. The top of stack pointer will hold the array index of the top of the stack. The stack will build up from the array indexed as one. An empty stack is indicated by having the top of stack pointer variable point to the zero index of the stack array variable. The bottom of the stack will actually be the one index of the stack array variable.

```
$ export DataStackPointer=0
$ declare -a DataStack; export DataStack
$ export ReturnStackPointer=0
$ declare -a ReturnStack; export ReturnStack
$
```

That's it for the required variables! And, yes, those could all be placed on one line, but I spread them over four lines for additional clarity.

functions

These are the functions for a Forth-like working environment. Not all of these functions exist in a real Forth. Most of these functions are have modifications from standard Forth. Many standard Forth functions are missing.

In particular, this approach treats the stack as a stack of objects. No matter how much memory space an object takes up, it only takes up a single stack slot. In a real Forth, data is stored on the stack as raw binary strings, characters, or numbers and the programmer is responsble for keeping track of how much memory every item actually uses.

ShowDataStack

The `ShowDataStack` function shows the current contents of the data stack in the order of top of stack to bottom of stack. This function is built for debugging purposes, but I am making it available for anyone to use.

```
$ ShowDataStack () {
>    echo "stack size of " $DataStackPointer;
>    if [ "$DataStackPointer" -gt "0" ] ; then
>        loopcounter=$DataStackPointer
>        while [ $loopcounter -gt 0 ] ;
>        do
>            echo ${DataStack["$loopcounter"]}
>            let "loopcounter=$loopcounter - 1"
>        done
>    else
>        echo "the stack is empty"
>    fi
>    }
$
```

push

The `push` function pushes a single object onto the data stack. Forth doesn’t have this word because simply typing a number is sufficient to push the number onto the stack. This function will be used by other functions to manipulate the data stack.

NOTE: This fucntion does *not* yet have error checking.

```
$ push () {
> let "DataStackPointer=$DataStackPointer + 1" #preincrement
> DataStack["$DataStackPointer"]=$1 #push data item on top of stack
> }
$
```

pop

The `pop` function pops a single object from the data stack. Forth has a different function for this functionality, but this function is intended for internal use by other functions to manipulate the data stack.

NOTE: This fucntion does *not* yet have error checking.

```
$ pop () {
> echo ${DataStack["$DataStackPointer"]} #pop data item from top of stack
> let "DataStackPointer=$DataStackPointer - 1" #postdecrement
> }
$
```

UNIX and Linux System Administration and Shell Programming

version history

For the most up to date version, visit the website http://www.osdata.com/

22 September 3, 2012. (Forth-like routines)
21 September 2, 2012. (cd)
20 September 2, 2012. (built-in commands; computer basics)
19 September 1, 2012. (cool shell tricks)
18 September 1, 2012. (cool shell tricks)
17 August 31, 2012. (cool shell tricks)
16 August 30, 2012. (defaults)
15 August 30, 2012. (less)
14 August 29, 2012. (history)
13 August 28, 2012. (less)
12 August 27, 2012. (test bed; command separator; installing software from source)
11 August 26, 2012. (quick tour)
10 August 25, 2012. (root, sudo)
9 August 25, 2012. (quick tour, shred, init, command structure)
8 August 24, 2012. (sudo, screencapture)
7 August 24, 2012. (tar, command structure)
6 August 23, 2012. (shells)
5 August 22, 2012. (passwd; cat; file system basics)
4 August 22, 2012. (login/logout; man)
3 August 21, 2012. (login/logout)
2 August 21, 2012. (shell basics)
1 August 19, 2012. (imported amd formatted already existing work from the website)

This book includes material from the http://www.osdata.com/ website and the text book on computer programming.