Linux Basics

An Introductory Exploration for those wishing to understand the Linux Operating System

Neale Ferguson
R&D Fellow

Disclaimer

References in this manual to Software AG products, programs, or services do not imply that Software AG, Inc. intends to make these available in all countries in which Software AG, Inc. operates.

Use, duplication, or disclosure by the Government of this commercial software as defined l clause 252.227-70414(a)(1) of the DFARS is subject to restriction and shall be deemed restricted computer software as defined in clause 52.227-19 of the FAR.

Printed in the United States of America.

The status symbols r and ?, as used to identify Software AG trademarks herein, refer to the status of Software AG trademarks as pending or registered in the U.S. Patent and Trademark Office. Software AG and/or its subsidiaries have applied for and have been granted registrations for their trademarks throughout the world. Software AG will act to enforce its trademark rights worldwide.

IBM is a registered trademark of International Business Machines Corporation.

Linux is a registered trademark of Linus Torvalds.

ADABAS, Natural, Tamino, Bolero, and EntireX are trademarks of Sterling Software, Inc., and/or its subsidiaries.

Class Agenda…

- Two parts of class
  - Part 1
    - Linux Concepts
    - Getting Started
    - Daemons
    - File Systems

Class Agenda

- Part 2
  - Accessing Your Data
  - vi – The System Editor
  - Self-study
    - *bash – The Scripting Language*

The Linux Kernel

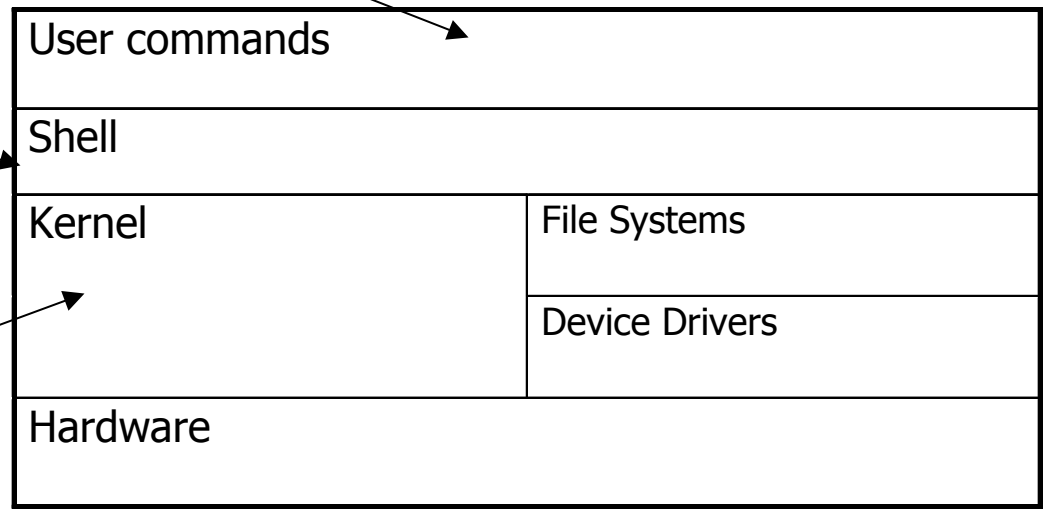
A quick look under the covers

The Linux System

User commands includes executable programs and scripts

The shell interprets user commands. It is responsible for finding the commands and starting their execution. Several different shells are available. Bash is popular.

The kernel manages the hardware resources for the rest of the system.



The Kernel Layer

- Basic Operating System
- Device support
- Memory Management
- Process Management
- Interface to the hardware
- A set of APIs
- TCP/IP integrated into kernel

Device Layer

- Exploits API from kernel
- Register driver with kernel
- Handle I/O requests for “type” of device
- Examples:
  - DASD
  - VDU
  - Tape

File Systems

- An layer of abstraction between underlying file scheme and device(s)
- VFS provides a single API between user and file system
- Handles “mounting”, I/O requests that get implemented (eventually) by a device driver

Shells

- Interface between user and kernel
- Can be more than one
- User can swap between them
- Command line and GUI
- More later…

Booting the Operating System

- Bootstrap read from initial medium
- Loads kernel
- Passes control to initialization
- Memory and I/O setup
- 1st process “init” started: all other processes are descendants of this one
- Invokes a shell
- Begins startup processes

```
IPL 192 CLEAR
Linux version 2.4.3-0.4.25vrdr (root@z02.millenux.com) (gcc
version 2.95.2 19991
024 (release)) #1 SMP Wed Jun 6 21:15:45 CEST 2001
Command line is: root=/dev/dasda1 ro dasd=192-193 noinitrd

We are running under VM
On node 0 totalpages: 65536
zone(0): 65536 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/dasda1 ro dasd=192-193
noinitrd

Highest subchannel number detected (hex) : 000C
Calibrating delay loop... 851.96 BogoMIPS
Memory: 241528k/262144k available (1399k kernel code, 20616k
reserved, 652k data, 60k init)
```

```
Detected 1 CPU's
Boot cpu address  0
cpu 0 phys_idx=0 vers=FF ident=087100 machine=2064 unused=0000
init_mach : starting machine check handler
init_mach : machine check buffer : head = 001F7850
init_mach : machine check buffer : tail = 001F7858
init_mach : machine check buffer : free = 001F7860
init_mach : CRW entry buffer anchor = 001F7868
init_mach : machine check handler ready
dasd: Registered successfully to major no 94
dasd(eckd): ECKD discipline initializing
dasd(eckd): We are interested in: CU 3990/00
dasd(eckd): We are interested in: CU 2105/00
dasd(eckd): We are interested in: CU 9343/00
dasd: Registered ECKD discipline successfully
dasd(fba):FBA  discipline initializing
dasd(fba):We are interested in: CU 6310/00
dasd(fba):We are interested in: CU 3880/00
dasd: Registered FBA discipline successfully
dasd(eckd): 0192 on sch 1: 3390/0A(CU:3990/01) Cyl:2838
Head:15 Sec:224
```

```
 INIT: version 2.78 booting

                         Welcome to Think Blue Linux
Mounting proc filesystem:  [  OK  ]
Configuring kernel parameters:  [  OK  ]
Setting clock : Mon Jul  9 16:20:12 EDT 2001 [  OK  ]
Activating swap partitions:  [  OK  ]
Setting hostname dali008.software-ag.de:  [  OK  ]
Checking root filesystem
/: clean, 40490/255488 files, 174547/510837 blocks

[/sbin/fsck.ext2 -- /] fsck.ext2 -a /dev/dasda1
Starting sendmail: [  OK  ]
Starting console mouse services: (no mouse is configured)
Starting crond: [  OK  ]
Starting xfs: [  OK  ]
Starting anacron: [  OK  ]
Think Blue Linux release 7.1 (verdigris)
Kernel 2.4.3-0.4.25vrdr on a s390x

dali008 login:
```

Introduction to Linux

Basic Concepts

Users and Groups

Users are identified by user identifications (UIDs), each of which is associated with an integer in the range of 0 to 4 294 967 295 (X'FFFFFFFF'). Users with UID=0 are given *superuser* privileges.

Users are placed in groups, identified by group identifications (GIDs). Each GID is associated with an integer in the range from 0 to 4 294 967 295

Let the system assign UID to avoid duplicates

Use `id` to display your user and group information

```
uid=500(neale) gid=500(neale) groups=500(neale),3(sys),4(adm)
```

Users and Groups

- Groups define functional areas/responsibilities
- They allow a collection of users to share files
- A user can belong to multiple groups
- You can see what groups you belong to using the **`groups`** command:

```
neale sys adm
```

Group Setup

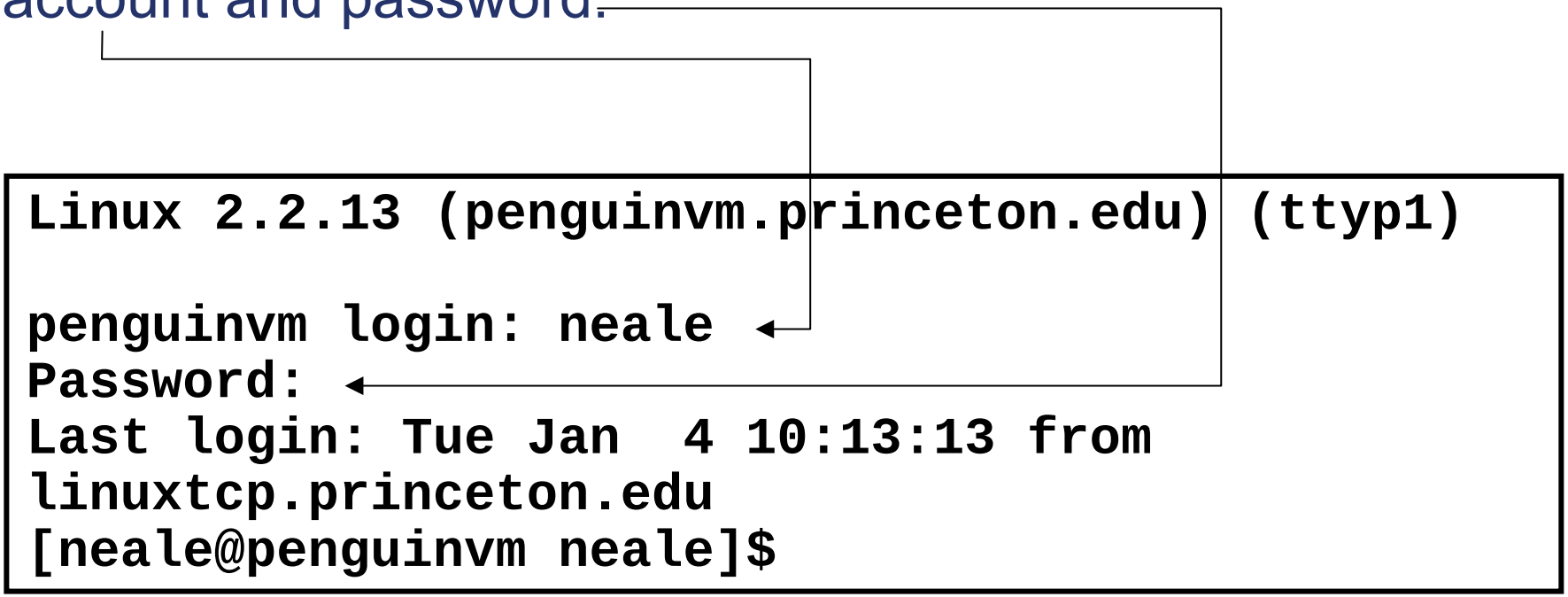
- Typical
  - sys
  - bin
  - adm
  - staff
  - users
- Software AG
  - odessy
  - adabasd
  - peport
  - pcc
  - intprod
  - network

Logging In

- Connect to the Linux system using telnet:
  - vt100, vt220, vt320
  - ansi
  - xterm
  - X-windows
- Able to login more than once with same user
- No ‘MW’ problems!

Logging In

- Before you can use it you must login by specifying your account and password:



Rule Number 1

- Do not login as root unless you have to
- root is the superuser
  - Protection mechanisms can be overridden
  - Careless use can cause damage
  - Has access to everything by default
- root is only user defined when you install
  - First thing is to change root's password
  - The second job is to define "normal" users for everyday use
- Use the **su** command to switch users to root
- Use **sudo** command to issue privileged commands

Creating a new user

- Use the **useradd**/**adduser** command
- Use the **passwd** command to set password

```
[root@penguinvm]# useradd scully
[root@penguinvm]# passwd scully
Changing password for user scully
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated
successfully
[root@penguinvm]#
```

Adding a new user

- Limits on users can be controlled by
  - Quotas
  - **`ulimit`** command
- Authority levels for a user controlled by group membership

Adding a New User

- Writes a new entry in **`/etc/passwd`**
- Also in **`/etc/shadow`**
- Why?
  - For security reasons
  - Explanation when we get to the section on files

Lab One

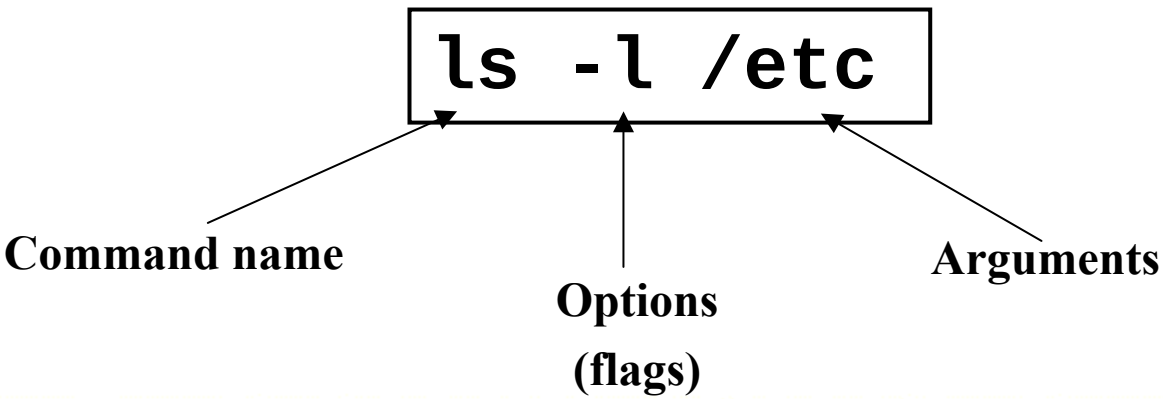
- Use telnet to connect to the lab machine
- Login using ID supplied
  - Userid student*nn* where *nn* = 01-15
  - Password: linx101 -- PLEASE DO NOT CHANGE IT!
- Logout using the **`exit`** or **`logout`** command

Introduction to Linux

Command Basics

Linux Command Basics

- To execute a command, type its name and arguments at the command line



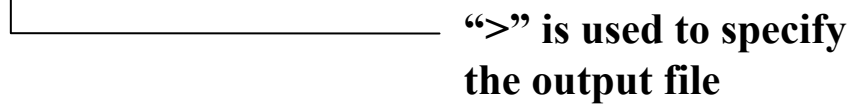
Standard Files

- UNIX concept of “standard files”
  - standard input (where a command gets its input) - default is the terminal
  - standard output (where a command writes it output) - default is the terminal
  - standard error (where a command writes error messages) - default is the terminal

Redirecting Output

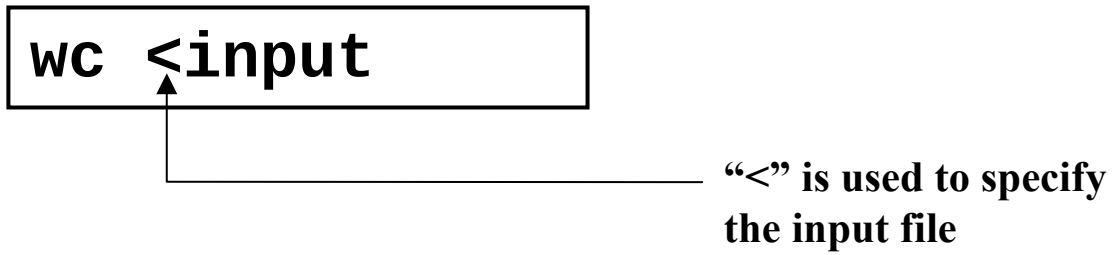
- The output of a command may be sent to a file:

```
ls -l >output
```



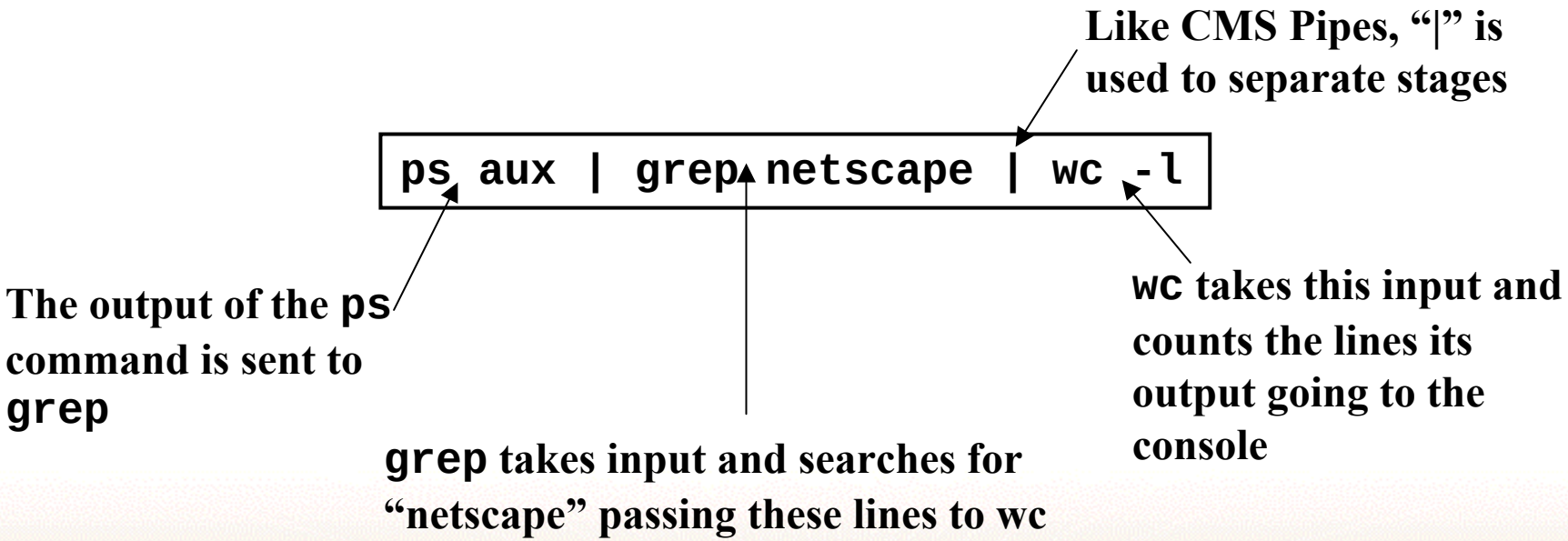
Redirecting Input

- The input of a command may come from a file:



Connecting commands with Pipes

- Not as powerful as CMS/TSO Pipes but the same principle
- The output of one command can become the input of another:



Command Options

- Command options allow you to control a command to a certain degree
- Conventions:
  - Usually being with a single dash and are a single letter (“`-l`”)
  - Sometimes have double dashes followed by a keyword (“`--help`”)
  - Sometimes follow no pattern at all

You need help?

- The Linux equivalent of HELP is **man** (manual)
  - Use **man -k <keyword>** to find all commands with that keyword
  - Use **man <command>** to display help for that command
    - Output is presented a page at a time. Use **b** for to scroll backward, **f** or a space to scroll forward and **q** to quit

Common Commands

- **`pwd`** - print (display) the working directory
- **`cd <dir>`** - change the current working directory to *dir*
- **`ls`** - list the files in the current working directory
- **`ls -l`** - list the files in the current working directory in long format
- **`shutdown –[hr] [now|time] [message]`**
  - Shutdown or restart the system

More Commands

- **`who`** or **`w`**
  - List who is currently logged on to the system
- **`whoami`**
  - Report what user you are logged on as
- **`ps`**
  - List your processes on the system
- **`ps aux`**
  - List all the processes on the system
- **`echo “A string to be echoed”`**
  - Echo a string (or list of arguments) to the terminal

Who's Logged On Right Now?

- The **w** command lists all users logged on right now

```
 5:16pm  up 2 days,  8:46,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
neale    ttyp0    websurfer.reston  4:28pm  1.00s  0.52s  0.18s  w
```

Lab Two

- Logon to your test machine
  - Get help on the **`ls`** command
  - Find out who else is on the system
  - What is your current directory
  - Redirect the output of the **`ls -l /`** command to **`ls.output`** and see what you get
  - Logout

Introduction to Linux

Daemons

Agenda

- What are Daemons?
- Common Daemons
- Additional Daemons

The Daemon Concept

- Daemons provide functions that are not available in the base operating system
- Comparable to
  - Services in NT
  - Service Virtual Machines in VM
  - Started tasks and built-in subsystems in OS/390
- Listen for work requests
- Perform service then disconnect

Common Daemons

- Apache - httpd
- LDAP - sldapd
- DNS - bind
- sendmail
- Samba - smbd/nmbd
- FTP - ftpd
- Usenet - innd
- Superdaemon - inetd

Apache

- World's most popular web server
- Version 1.3.14 most current
- Version 2.0 Alpha just released

LDAP

- Lightweight Directory Access Protocol
- Based on entries which are collections of attributes that have a name (a distinguished name)
- Entries are arranged in a hierarchical tree-like structure
- LDAP defines operations for interrogating and updating the directory

DNS

- Domain Name Server
- Resolves IP names to IP addresses (and vice versa)
- Forwards on requests it cannot resolve
- Fields requests from within and without host

Samba

- A collection of programs that implement the Server Message Block (SMB) protocol for UNIX systems
- File and print serving
- NetBIOS name serving and browser support
- Support utilities

Samba

- Why?
  - Integrate Microsoft or IBM style desktop machines with Enterprise servers
  - Integrate Microsoft servers with Enterprise servers
  - Replace multiple protocols (e.g. DecNet, Novell NCP)

Samba

- What can it do?
  - Windows NT and LAN manager style file and print services to clients
  - A NetBIOS nameserver which provides browsing support (Samba can be the master browser)
  - FTP-like SMB client so you can access PC resources from VM
  - A limited command-line tool that supports some NT administrative functions

INND - Usenet

- A highly used and highly visible feature of the Internet
- Conduct discussions and disseminate them to interested parties
- Ported and configured INND-1.5.1 as part of the Residency

INETD

- INETD
  - Internet Super Daemon
  - Automatically starts other daemons upon request from client
  - Can be used to start Samba, Apache, Daytime
  - Can have multiple INET daemons
  - Also has internal services
    - chargen
    - discard
    - echo

Lab Three

- Telnet and Login to ID
- **`ps -ef | more`** -- Do you see any of the daemons we've talked about?
  - httpd
  - inetd
- Logout

Introduction to Linux

The Linux File Systems

About the Linux File Systems

- Linux files reside on:
  - Fullpack DASD
  - Minidisks
  - SCSI!
  - Partitions of any of the above
- Linux supports multiple file systems:
  - extfs2
  - fat/vfat
  - hpfs
  - jfs

Linux Device Handling

- Devices are the way Linux talks to the world
- Devices are special files in the **`/dev`** directory (try **`ls /dev`**)

`/dev/ttyx`	TTY devices
`/dev/hdb`	IDE hard drive
`/dev/hdb1`	Partition 1 on the IDE hard drive
`/dev/dasda`	ECKD/CKD/FBA DASD
`/dev/dasda1`	Partition 1 on DASD
`/dev/null`	The null device (“hole”)
`/dev/zero`	An endless stream of zeroes
`/dev/mouse`	Mouse (not /390)

Devices and Drivers

- Each **/dev** file has a major and minor number
  - Major defines the device type
  - Minor defines device within that type
  - Drivers register a device type

```
brw-r--r--   1 root     root       64,   0 Jun  1  1999 /dev/mnda
crw-r--r--   1 root     root        5,   0 Jan  5 09:18 /dev/tty
```

Special Files - /proc

- Information about internal Linux processes are accessible to users via the **`/proc`** file system (in memory)

/proc/cpuinfo	CPU Information
/proc/interrupts	Interrupt usage
/proc/version	Kernel version
/proc/modules	Active modules

```
cat /proc/cpuinfo
vendor_id       : IBM/S390
# processors    : 1
bogomips per cpu: 86.83
processor 0: version = FF, identification = 045226, machine = 9672
```

File Systems

- Linux supports many different types
- Most commonly, ext2fs
  - Filenames of 255 characters
  - File sizes up to 2GB
  - Theoretical limit 4TB
- Derived from extfs
- Highly reliable and high performer

File Systems

- Other file systems:
  - sysv - SCO/Xenix
  - ufs - SunOS/BSD
  - vfat - Win9x
  - msdos - MS-DOS/Win
  - umsdos - Linux/DOS
  - ntfs - WinNT (r/o)
  - hpfs - OS/2
  - cms - CMS (r/o)
- Other File systems:
  - iso9660 (CD-ROM)
  - nfs - NFS
  - coda - NFS-like
  - ncp - Novell
  - smb - LANManager
  - afs - Andrew File System

File Systems

- **mount**
  - Mounts a file system that lives on a device to the main file tree
  - Start at Root file system
    - Mount to root
    - Mount to points currently defined to root
  - **/etc/fstab** used to establish boot time mounting

```
/dev/dasda1     /              ext2    defaults,errors=remount-ro 0 1
/dev/dasdb1     /bin           ext2    defaults,errors=remount-ro 0 1
/dev/dasdc1     /usr           ext2    defaults,errors=remount-ro 0 1
/dev/dasdd1     /usr/local     ext2    defaults,errors=remount-ro 0 1
/dev/dasde1     /usr/man       ext2    defaults,errors=remount-ro 0 1
/dev/dasdf1     /home          ext2    defaults,errors=remount-ro 0 1
/dev/dasdg1     swap           swap    defaults    0       0
none            /proc          proc    defaults    0       0
```

File Systems

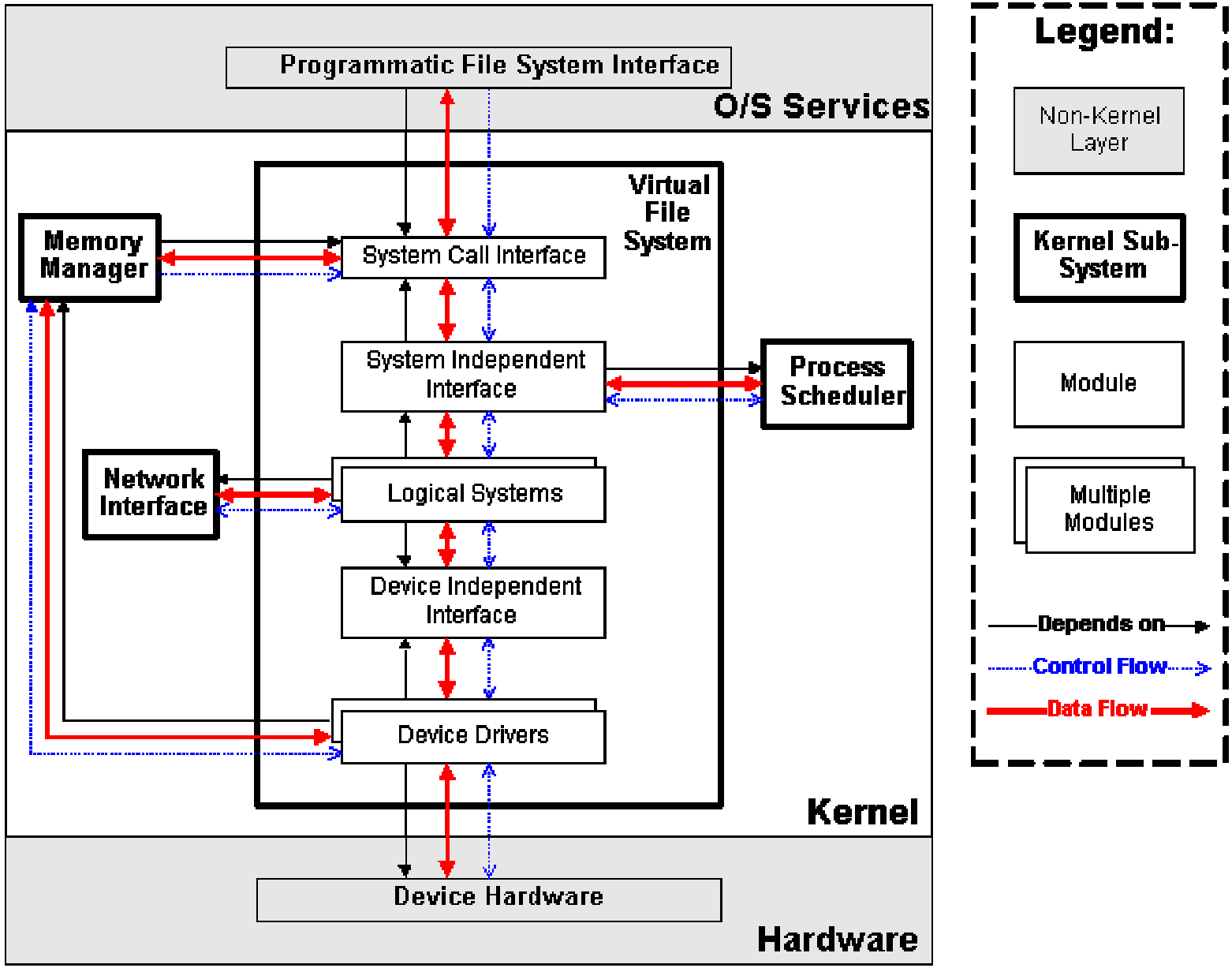
- You can view what file systems are mounted using either:
  - **mount**
  - **df**

Virtual File System

- VFS is designed to present a consistent view of data as stored on hardware
- Almost all hardware devices are represented using a generic interface
- VFS goes further, allowing the sysadmin to mount *any* of a set of logical file systems on *any* physical device

Virtual File System

- Analogous to CMS:
  - SFS
  - Minidisks
- Two different designs
- Common/transparent access

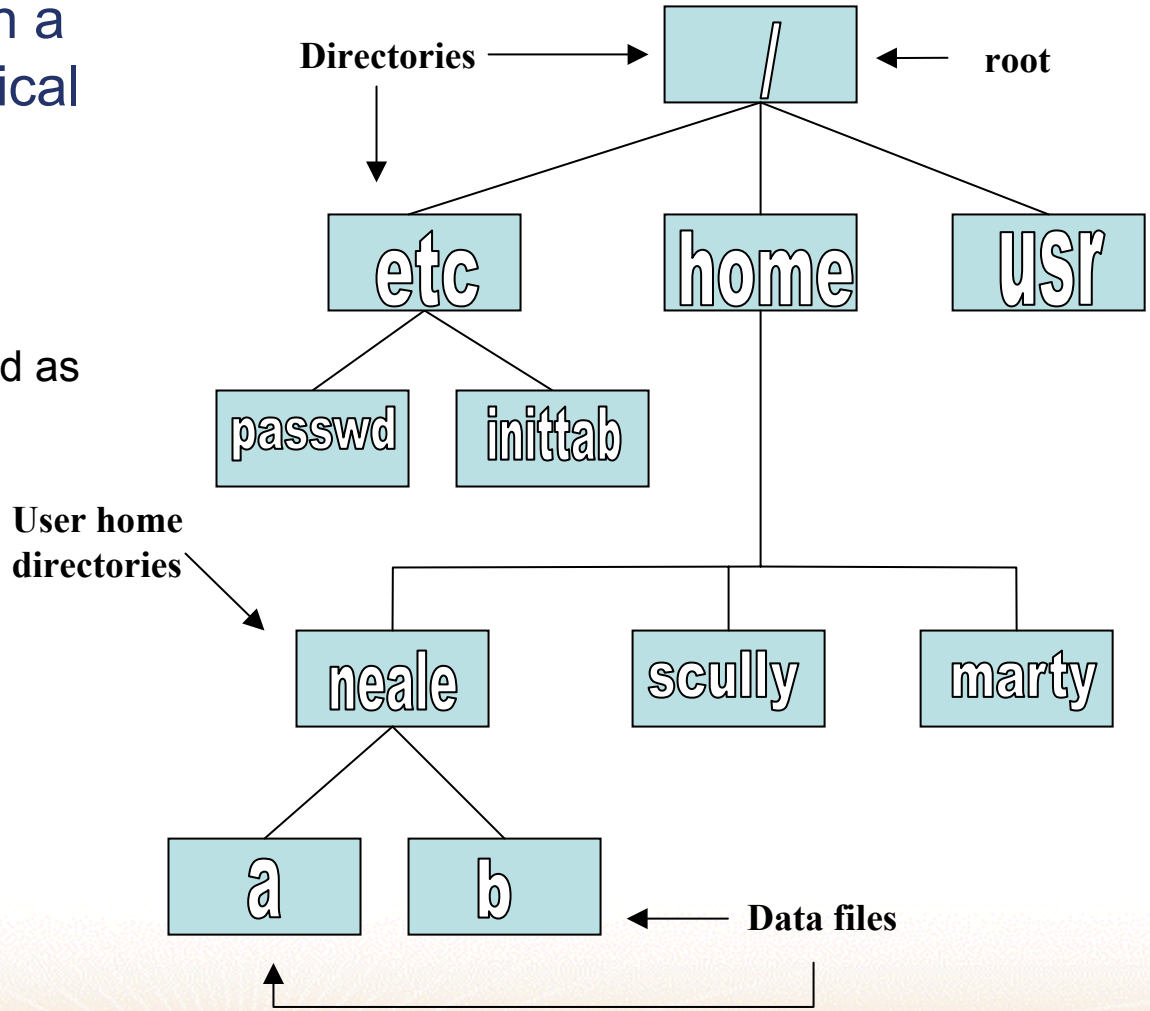

Programmatic File System Interface
O/S Services
Virtual File System
Memory Manager
System Call Interface
System Independent Interface
Process Scheduler
Network Interface
Logical Systems
Device Independent Interface
Device Drivers
Kernel
Device Hardware
Hardware
Legend:
Non-Kernel Layer
Kernel Sub-System
Module
Multiple Modules
Depends on
Control Flow
Data Flow

Lab Four

- Telnet and login to ID
- Find out what devices are mounted and what file systems are in use
- Examine a couple of the **`/proc`** files using the **`more`** command
- Logout

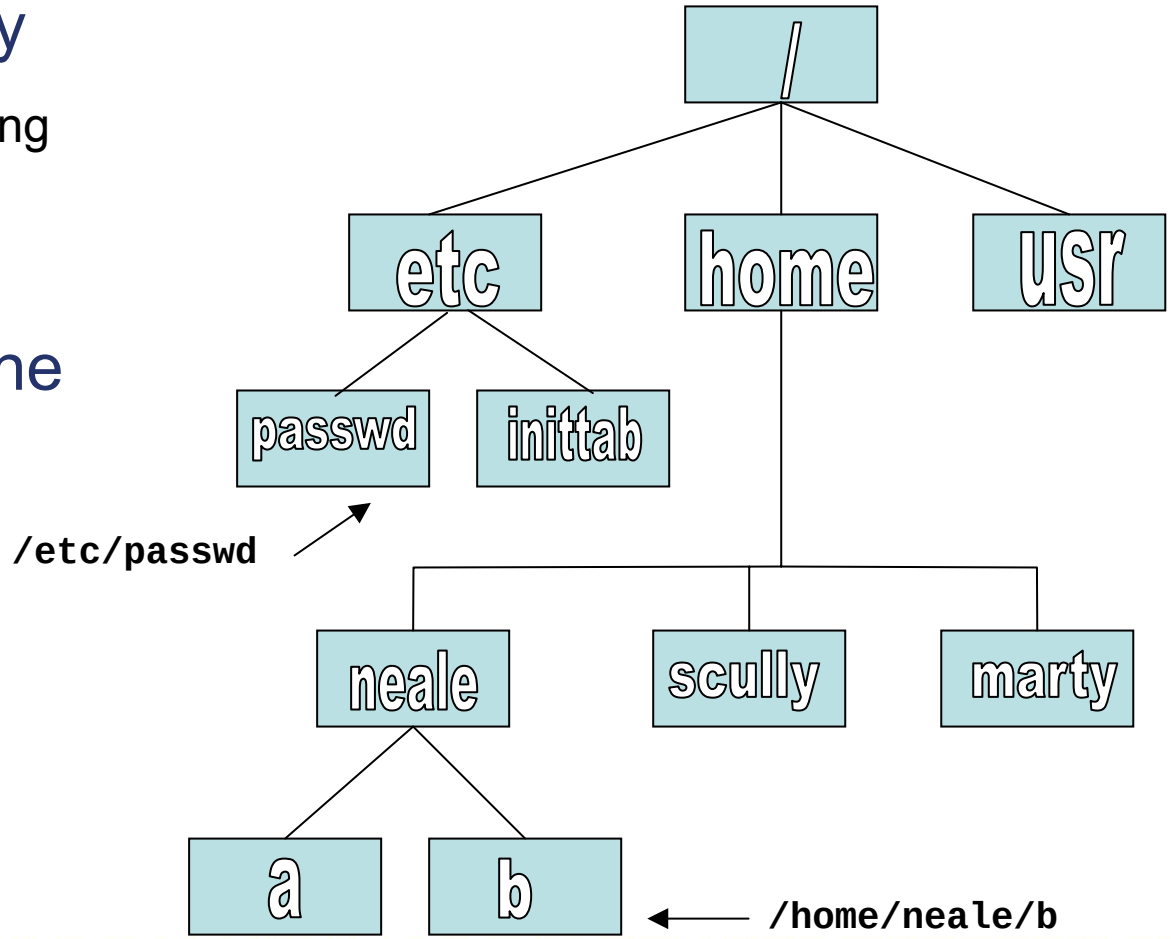
Linux File System Basics

- Linux files are stored in a single rooted, hierarchical file system
  - Data files are stored in directories (folders)
  - Directories may be nested as deep as needed



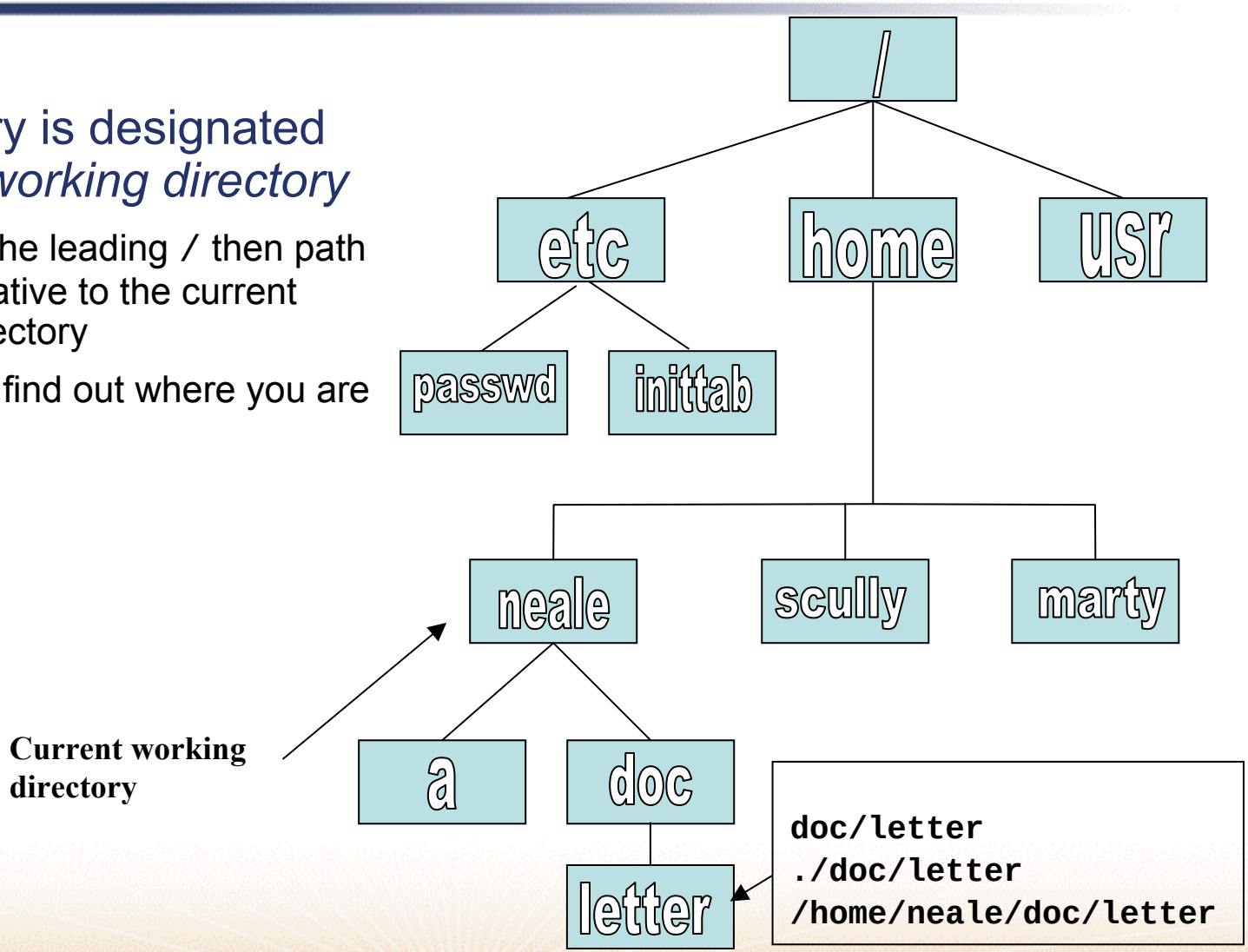
Naming Files

- Files are named by
  - naming each containing directory
  - starting at the root
- This is known as the *pathname*



The Current Directory

- One directory is designated the *current working directory*
  - if you omit the leading / then path name is relative to the current working directory
  - Use **pwd** to find out where you are



Some Special File Names

- Some file names are special:
  - **/** The root directory (don't confuse with the root user)
  - **.** The current directory
  - **..** The parent (previous) directory
  - **~** My home directory
  - ***~jane*** Jane's home directory
- Examples:
  - **./a** same as **a**
  - **../jane/x** go up one level then look in directory **jane** for **x**

Special Files

- **`/home`** - all users' home directories are stored here
- **`/bin`**, **`/usr/bin`** - system commands
- **`/sbin`**, **`/usr/sbin`** - commands used by sysadmins
- **`/etc`** - all sorts of configuration files
- **`/var`** - logs, spool directories etc.
- **`/dev`** - device files
- **`/proc`** - special system files

Lab Five

- Explore the file system
  - Use the **cd** command to go the “root” of the file system
  - Use **ls** to list the files and directories
  - Use the **cd** command to go to your home directory
  - Use the **pwd** command to display the name of the present working directory

Creating Files and Directories

- Files can be created in a number of ways
  - The output of a command
  - Being edited using vi or your favorite editor
  - By using the **touch** command which creates an empty file or updates the modification and access time information of an existing file
- Directories are created using the **mkdir** command

File Permissions

- Every file:
  - Is owned by someone
  - Belongs to a group
  - Has certain access permissions for owner, group, and others
  - Default permissions determined by **`umask`**

File Permissions

- Every user:
  - Has a uid (login name), gid (login group) and membership of a "groups" list:
    - The *uid* is who you are (name and number)
    - The *gid* is your initial “login group” you normally belong to
    - The *groups list* is the file groups you can access via group permissions

File Permissions

- Linux provides three kinds of permissions:
  - Read - users with read permission may read the file or list the directory
  - Write - users with write permission may write to the file or new files to the directory
  - Execute - users with execute permission may execute the file or lookup a specific file within a directory

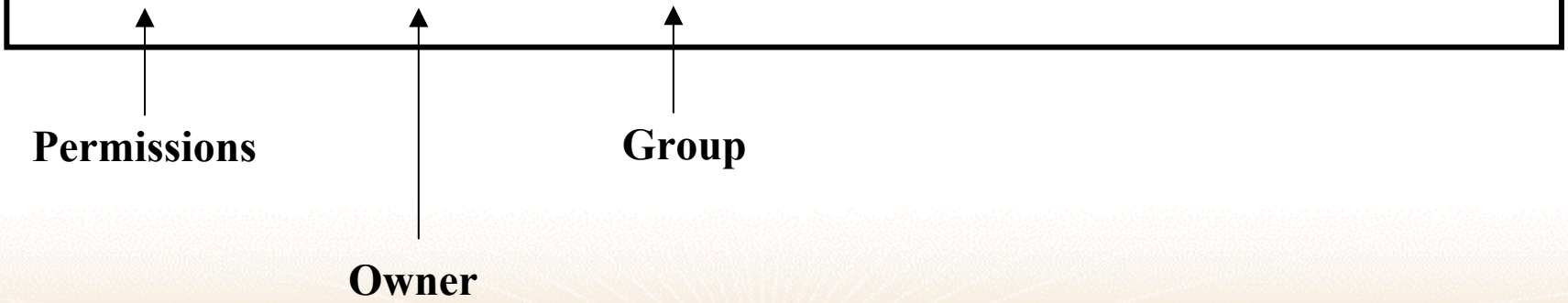
File Permissions

- Under MS-DOS, Windows, OS/2
  - File extensions determine if a file is “executable”
  - Uses .EXE .CMD .BAT
- UNIX/Linux
  - File privileges determine if a file should be executed
  - Contents of header or 1st line of file tell system how to execute

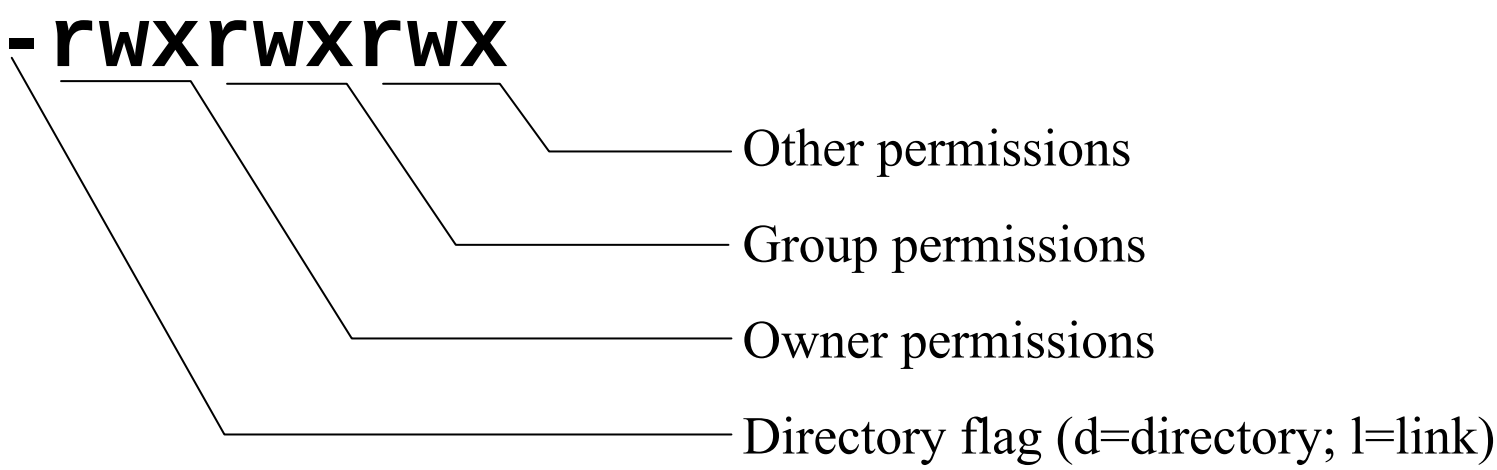
File Permissions

- The long version of a file listing (**`ls -l`**) will display the file permissions:

```
-rwxrwxr-x   1 rvdheij  rvdheij      5224 Dec 30 03:22 hello
-rw-rw-r--   1 rvdheij  rvdheij       221 Dec 30 03:59 hello.c
-rw-rw-r--   1 rvdheij  rvdheij      1514 Dec 30 03:59 hello.s
drwxrwxr-x   7 rvdheij  rvdheij      1024 Dec 31 14:52 posixuft
```



Interpreting File Permissions



Changing File Permissions

- Use the **chmod** command to change file permissions
  - The permissions are encoded as an octal number

User			Group			Other		
Read r	Write w	Execute x	Read r	Write w	Execute x	Read r	Write w	Execute x
400	200	100	40	20	10	4	2	1

```
chmod 0755 file  # Owner=rwx Group=r-x Other=r-x
chmod 0500 file2 # Owner=r-x Group=--- Other=---
chmod 0644 file3 # Owner=rw- Group=r-- Other=r--

chmod +x  file   # Add execute permission to file for all
chmod o-r file   # Remove read permission for others
chmod a+w file   # Add write permission for everyone
```

Remember `/etc/passwd`?

- Originally file permissions allowed “world read”
- Weakly encrypted passwords could be read by anyone!!
- **`/etc/shadow`** implemented with stricter permissions and stronger encrypting

```
[usanefe@dali157 - usanefe] ls -l /etc/passwd /etc/shadow
-rw-r--r--    1 root     root         2985 Jul  6 18:16 /etc/passwd
-rw-r-----    1 root     shadow       1468 Jul  7 13:32 /etc/shadow
```

Links?

- Links are references to files (aliases)
- Two forms:
  - Hard
  - Symbolic
    - Can point to files on different physical devices
    - Delete of original leaves link / Delete of link leaves original
    - Can be created for directories
- Create using **`ln`** or **`ln -s`** command
- The **`ls -l`** command will show you the links:

```
train01@reslx390:~ > ls -l /lib
total 10780
-rwxr-xr-x   1 root     root       367598 Nov  3  2000 ld-2.1.3.so
lrwxrwxrwx   1 root     root           11 Nov 29  2000 ld.so.1 -> ld-2.1.3.so
-rwxr-xr-x   1 root     root        21498 Nov  3  2000 libBrokenLocale.so.1
```

Lab Six

- Explore your filesystem:
  - Identify 1st level directories
  - Locate a symbolic link
  - Use the **`umask`** command to display current default
- Create 3 files (‘`all`’, ‘`group`’, ‘`owner`’) & assign permissions:
  - `all` - r/w to owner, group, and others
  - `group` - r/w to owner and group, r/o to others
  - `owner` - r/w to owner, r/o to group, none to others
- Create a directory ‘`test`’ under your home directory
  - Create a file ‘`real.file`’
  - Create a symbolic link in your home directory to ‘`real.file`’ called ‘`symbolic.link`’

Questions and Answers

Class Agenda -- Part 2

- Accessing Your Data
- vi – The System Editor
- the – XEDIT/ISPF clone
- bash – The Scripting Language

Shells

- An interface between the Linux system and the user
- Used to call commands and programs
- An interpreter
- Powerful programming language
  - “Shell scripts” = .bat .cmd EXEC REXX

Shells

- **sh** Bourne shell - the original
- **csh** C shell - compatible with Bourne shell
- **bash** Bourne again shell - most common on Linux
- **tcsh** The enhanced C shell
- **zsh** Z shell - newest, compatible with Bourne shell
- **ksh** Korn shell - most popular UNIX shell

Another definition of a Shell

- A shell is any program that takes input from the user, translates it into instructions that the operating system can understand, and conveys the operating system's output back to the user.
  - i.e. Any User Interface
  - Character Based v Graphics Based

Why Do I Care About The Shell?

- Shell is Not an Integral Part of O/S
  - UNIX Among First to Separate
  - Compare to MS-DOS, Mac, Win95, VM/CMS
  - GUI is NOT Required
  - Default Shell Can Be Configured
    - `chsh -s /bin/bash`
    - `/etc/passwd`
  - Helps To Customize Environment

Using the Shell

- Useful keys:
  - Cursor arrows:
    - Up/down - scroll through previous commands
    - Left/right - move over characters within the command line
    - Backspace/Delete - delete character
  - Control characters
    - CTRL-C - Abort command
    - CTRL-U- Delete the whole line
    - CTRL-Z - Suspend current process
    - CTRL-T - Swap current and next characters in command line
- Shortcuts
  - Word completion: Press TAB key to have Shell complete the line for you

Lab Seven

- Using the Shell
  - What shell are you using:
  - Editing the command line:
    - Scrolling through past commands
    - Inserting/deleting characters on command line
    - Using editing key: CTRL-T
    - Try command completion. What happens when:

      `ls /etc/pro<TAB>`
  - Invoke the C shell

Shell Scripts

```
#!/bin/bash
while
true
do
   cat somefile > /dev/null
   echo .
done
```

```
/* */
do forever
   'PIPE < SOME FILE | hole'
   say '.'
end
```

Filename Expansion

- Shell will scan for special characters
- Process called “globbing”
- Not the same as regular expressions
- Performs expansion:
  - **`ls *.c`** List all files with extension of ‘c’
  - **`ls *.[ch]`** List all files with extension of ‘c’ or ‘h’
  - **`ls *[0-9]*.c`** List all files with extension of ‘c’ with a name consisting of 0 or more numeric characters
  - **`ls ab?de.c`** List all files with extension of ‘c’ whose first two letter of the file name are “ab” and last two letters are “de”

Switching Users

- **su <*accountname*>**
  - switch user accounts. You will be prompted for a password. When this command completes, you will be logged into the new account. Type **exit** to return to the previous account
- **su**
  - Switch to the root user account. Do not do this lightly

Note: The root user does not need to enter a password when switching users. It may become any user desired. This is part of the power of the root account.

Environment Variables

- Environment variables are global settings that control the function of the shell and other Linux programs. They are sometimes referred to global shell variables.
- Setting:
  - `VAR=/home/fred/doc`
  - `export TERM=ansi`
  - `SYSTEMNAME=\`uname -n\``
- Similar to GLOBALV SET … in CMS

Environment Variables

- Using Environment Variables:
  - **`echo $VAR`**
  - **`cd $VAR`**
  - **`cd $HOME`**
  - **`echo “You are running on $SYSTEMNAME”`**
- Displaying - use the following commands:
  - **`set`** (displays local & environment variables)
  - **`export`**
- Variables can be retrieved by a script or a program

Some Important Environment Variables

- HOME
  - Your home directory (often be abbreviated as “~”)
- TERM
  - The type of terminal you are running (for example vt100, xterm, and ansi)
- PWD
  - Current working directory
- PATH
  - List of directories to search for commands

PATH Environment Variable

- Controls where commands are found
  - PATH is a list of directory pathnames separated by colons. For example:

    `PATH=/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/home/scully/bin`
  - If a command does not contain a slash, the shell tries finding the command in each directory in PATH. The first match is the command that will run

PATH Environment Variable

- Similar to setting the CMS search order
- Usually set in **`/etc/profile`** (like the SYSPROF EXEC)
- Often modified in **`~/.profile`** or **`~/.bashrc`** or **`~/.login`** (like the PROFILE EXEC)

File Commands

- **`cp <fromfile> <tofile>`**
  - Copy from the <fromfile> to the <tofile>
- **`mv <fromfile> <tofile>`**
  - Move/rename the <fromfile> to the <tofile>
- **`rm <file>`**
  - Remove the file named <file>
- **`mkdir <newdir>`**
  - Make a new directory called <newdir>
- **`rmdir <dir>`**
  - Remove an (empty) directory

More Commands

- **`alias`** - used to tailor commands:
  - **`alias erase=rm`**
  - **`alias grep="grep -i"`**
- **`ar`** - Maintain archive libraries: a collection of files (usually object files which may be linked to a program, like a CMS TXTLIB)

```
ar -t libgdbm.a
__.SYMDEF
dbmopen.o
```

More Commands

- **awk** - a file processing language that is well suited to data manipulation and retrieval of information from text files
- **chown** - sets the user ID (UID) to owner for the files and directories named by pathname arguments. This command is useful when from test to production

```
chown -R apache:httpd /usr/local/apache
```

More Commands

- **`diff`** - attempts to determine the minimal set of changes needed to convert a file specified by the first argument into the file specified by the second argument
- **`find`** - Searches a given file hierarchy specified by path, finding files that match the criteria given by expression

More Commands

- **grep** - Searches files for one or more pattern arguments. It does plain string, basic regular expression, and extended regular expression searching

```
find ./ -name "*.c" | xargs grep -i "fork"
```

In this example, we look for files with an extension "c" (that is, C source files). The filenames we find are passed to the xargs command which takes these names and constructs a command line of the form: `grep -i fork <file.1>…<file.n>`. This command will search the files for the occurrence of the string "fork". The "`-i`" flag makes the search case insensitve.

More Commands

- **kill** - sends a signal to a process or process group
- You can only kill your own processes unless you are root

```
UID         PID  PPID  C STIME TTY          TIME CMD
root       6715  6692  2 14:34 ttyp0    00:00:00 sleep 10h
root       6716  6692  0 14:34 ttyp0    00:00:00 ps -ef
[root@penguinvm log]# kill 6715
[1]+  Terminated              sleep 10h
```

More Commands

- **`make`** - helps you manage projects containing a set of interdependent files (e.g. a program with many source and object files; a document built from source files; macro files)
- **`make`** keeps all such files up to date with one another: If one file changes, **`make`** updates all the other files that depend on the changed file
- Roughly the equivalent of VMFBLD

More Commands

- **sed** - applies a set of editing subcommands contained in a script to each argument input file

```
find ./ -name "*.c,v" | sed 's/,v//g' | xargs grep "PATH"
```

This `find`s all files in the current and subsequent directories with an extension of c,v. `sed` then strips the ,v off the results of the find command. `xargs` then uses the results of `sed` and builds a `grep` command which searches for occurrences of the word PATH in the C source files.

More Commands

- **`tar`** - manipulates archives
  - An archive is a single file that contains the complete contents of a set of other files; an archive preserves the directory hierarchy that contained the original files.

```
tar -tzf imap-4.7.tar.gz
imap-4.7/
imap-4.7/src/
imap-4.7/src/c-client/
imap-4.7/src/c-client/env.h
imap-4.7/src/c-client/fs.h
```

Introduction to Linux

Accessing Your Data

Accessing Your Data

- Data files are accessed by pathname (relative or absolute)
- Command files are accessed via PATH environment variable
- System wide PATH set in **`/etc/profile`**
- User specific PATH may be set in **`~/.profile ~/.bashrc ~/.login`**

Listing Your Files

- The **`ls`** command is used for listing files and their attributes:
  - **`ls <pathname>`**
  - **`ls -l <pathname>`**
  - **`ls -la <pathname>`**

ls

```
[neale@penguinvm neale]$ ls /etc
DIR_COLORS       ftpusers       login.defs      quota.conf
DOMAINNAME       gettydefs      logrotate.d     rc.d
HOSTNAME         group          mail.rc         resolv.conf
HOSTNAME.orig    group-         man.config      resolv.old
X11              group.OLD      mime-magic      rpc
adjtime          group~         mime-magic.dat  security
aliases          host.conf      mime.types      sendmail.cf
aliases.db       hosts          motd            sendmail.st
aliases~         hosts.allow    mtab            services
bashrc           hosts.allow~   named.conf      shells
conf.linuxconf   hosts.deny     named.conf~     ssh_config
cron.d           hosts~         nscd.conf       ssh_host_key
cron.daily       httpd          nsswitch.conf   ssh_host_key.pub
cron.weekly      inetd.conf     nsswitch.conf~  ssh_random_seed
csh.login        inetd.conf~    pam.d           sshd_config
default          info-dir       passwd          sysconfig
exports          initlog.conf   passwd-         syslog.conf
fdprm            inittab        ppp             termcap
fstab            inputrc        printcap        zlogin
ftpaccess        ioctl.save     profile         zlogout
ftpconversions   ld.so.cache    profile.d       zprofile
ftpgroups        ld.so.conf     protocols       zshenv
ftphosts         localtime      pwdb.conf       zshrc
```

ls

- Color output?
  - **/etc/DIR_COLORS**

```
COLOR tty
# Below, there should be one TERM entry for each termtype that is colorizable
TERM linux
EIGHTBIT 1
# 00=none 01=bold 04=underscore 05=blink 07=reverse 08=concealed
# Text color codes:
# 30=black 31=red 32=green 33=yellow 34=blue 35=magenta 36=cyan 37=white
# Background color codes:
# 40=black 41=red 42=green 43=yellow 44=blue 45=magenta 46=cyan 47=white
NORMAL 00       # global default, although everything should be something.
FILE 00         # normal file
DIR 01;34       # directory
```

ls -l

- "DIR" like output:

```
[neale@penguinvm neale]$ ls -l
total 1612
-rw-r--r--   1 neale    neale      148119 Jan 14 10:12 %backup%~
-rw-------   1 neale    neale         511 Jan 18 10:58 Linux
drwxrwxr-x   7 neale    neale        1024 Mar 17 12:47 ORBit-0.5.1
drwxr-xr-x   7 neale    neale        1024 Mar 13 09:08 apache_2.0
-rw-rw-r--   1 neale    neale     1476724 Mar 11 22:18 apache_2.0a1.tar.gz
drwxrwxr-x   9 neale    neale        1024 Feb 14 20:58 classpath-0.00
-rw-rw-r--   1 neale    neale        1215 Jan 12 15:54 config.patch
drwxrwxr-x   2 neale    neale        1024 Mar 20 19:12 cpint
drwxrwxrwx   2 neale    develope     1024 Feb  9 11:26 html
-rw-r--r--   1 neale    neale         994 Feb 24 22:05 ip.num
-rw-rw-r--   1 neale    neale        1344 Feb 24 22:06 ip.num.sh
drwxrwxr-x  11 neale    neale        1024 Feb 25 21:08 japhar-0.08
drwxrwxr-x   5 neale    neale        1024 Jan 17 09:42 ltxml-1.1
-rw-rw-r--   1 neale    neale          81 Mar  7 17:57 test.c
-rwxrwxr-x   1 neale    neale         790 Mar  7 17:59 test.s
drwxrwxr-x   2 neale    neale        1024 Feb 29 15:13 tmp
```

ls -la

- List “hidden” files:

```
[neale@penguinvm neale]$ ls -la .*[a-zA-Z]
-rw-------   1 neale    neale         985 Mar 20 10:52 .Xauthority
-rw-------   1 neale    neale       15044 Mar 22 12:49 .bash_history
-rw-r--r--   1 neale    neale           6 Jan 18 10:58 .mailboxlist
-rw-rw-r--   1 neale    neale         153 Feb 23 14:17 .profile
-rw-rw-r--   1 neale    neale         250 Dec 31 12:04 .therc
```

Viewing Files

- **cat** "Concatenate"
- **more** Display one page at a time
- **less** Variant of **more**
- Editors
  - **vi** Visual editor, the default
  - **the** XEDIT/KEDIT/ISPF clone
  - **xedit** X windows text editor
  - **emacs** Extensible, Customizable Self-Documenting Display Editor
  - **pico** Simple display-oriented text editor
  - **nedit** X windows Motif text editor

cat

- Concatenate files and print on the standard output

```
[neale@penguinvm neale]$ cat .profile
alias dir="ls --color -laA"
alias ls="ls --color"
export PATH=./:/sbin:/usr/sbin:$PATH:/usr/local/japhar/bin
export JAPHAR_LOG="ALL,999,/tmp/japhar.log"
```

more

- File perusal filter for page-at-a-time viewing

```
[neale@penguinvm neale]$ more test.s
        .file   "test.c"
        .version        "01.01"
gcc2_compiled.:
.text
:
:
.L$CO1: AHI     13,.L$PG1-.L$CO1
        ST      0,0(15)
        LR      11,15
        LR    9,7
        ST    2,96(11)
--More--(71%)
```

Lab Eight

- Listing and displaying files
  - Use the **`ls -a`** command to display directories (where did all those files come from??)
  - Use the **`-R`** option of **`ls`** to display down file tree
  - Use **`cat`** to display a file
  - Use **`more`** to display a file one page at a time
  - Erase the link '`symbolic.link`', erase the '`test`' directory and its contents, then erase the '`all`', '`group`', and '`owner`' files.

Introduction to Linux

Editors

vi Basics...

'Editors are like religion; the one you grew up with is the only "true" one'

- **vi** was the first real screen-based editor for UNIX
- **vi** comes with every UNIX system
- **vi** may be invoked from the command line by typing the command followed by the file identifier of the file to be edited

```
vi <pathname>
```

vi Basics

- Pronounced: *vee-eye*
- When using **`vi`** you are in one of three modes:
  - Command mode: the mode you start in
  - Edit mode: allows you to do “editing”
  - Ex mode: where you communicate with **`vi`** to do things with the file
- Only a few things you *need* to know, lots of things you *could* know
- Best way to learn is by doing...

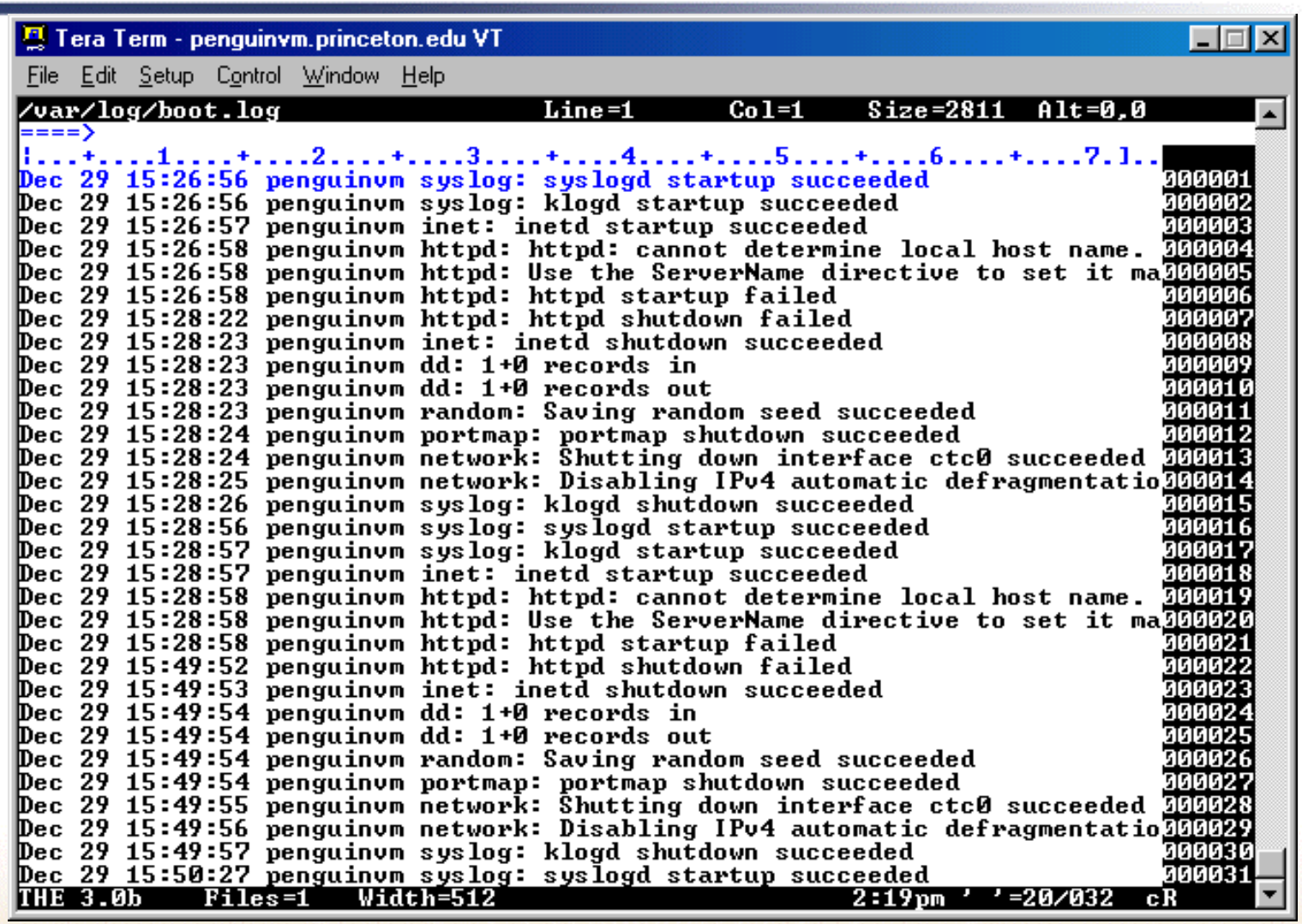
Lab Nine

- Use “vi Primer”
- Perform actions according to the guide

THE Basics

- The THE environment provides an additional set of commands oriented toward editing a file
  - An input area (command line) is provided for the entry of commands
  - Linux commands may be executed by prefacing them with `DOS`

Default Look of a THE Session



THE Commands: Things of Note

- The screen is considered a “window” on the file
- Movement commands (UP, DOWN, LEFT, RIGHT) describe movement of the *window* relative to the file
  - The command “down 6” moved the window down -- or forward -- 6 lines in the file
- Additional movement commands are available
  - TOP and BOTTOM move the window to the top or bottom of the file
  - Use ‘:n’ to request a particular line
  - The requested line is positioned on the “current line”

THE Prefix Commands

- In addition to the command line, you can also enter commands in the prefix area of a line
- Some common prefix commands include:
  - **I** - insert
  - **si** - insert a series of lines
  - **/** - make this the current line
  - **M** or **MM** - move a line, **M**, or a group of lines, **MM**
  - **C** or **CC** - copy a line, **C**, or a group of lines, **CC**
  - **P** - execute move or copy Preceding this line
  - **F** - execute more of copy Following this line

THE Input Area Commands

- **SET**
  - Change characteristics of your default view
  - Change characteristics of your file
- **Input** - Creates an input area for free form typing
- Scrolling and positioning commands
- **LOCATE** - find strings in the file
- **CHANGE** command - change commands in the file
- **SAVE** and **FILE**

THE Macros

- Create your own **`.therc`** to customize your view of **`the`**
  - Color (if available)
  - Placement of items discussed
    - scale
    - messages
    - command line, etc.
  - Autosave frequency
- **`the`** macros are REXX (Regina) programs that run in the **`the`** environment to perform specific tasks

This Looks Like the ISPF Editor

- The editors do share many characteristics
- There's just enough similarity to lull you into a false sense that you know what you're doing. E.g.
  - The biggest area of conflict/confusion is prefix commands
    - 'A' in THE is "add a line following this one"
    - 'A' in ISPF is a target for moving or copying lines ("move/copy the lines after this one")
    - The THE equivalent of ISPF's 'A' prefix command is the 'F' prefix command ("move or copy following this line")
  - "Insert mode" (for adding multiple lines to a file) works very differently in the two environments

THE Exercises…

- Edit the file **`the.sample`**
- Insert a line at the top of the file and type your name
- Copy that line to the bottom of the file
- Move the 2nd paragraph behind the 3rd paragraph
- Split the first line of the first paragraph before the word ‘honorably,’
- Join the 4th line to the new 3rd line new text after the word on that line
- Duplicate the 2nd line with your name 8 times
- File the file when you are done

…THE Exercises

- Edit the file **`~/.therc`**
- Change the prefix area to numbers with no leading zeros
- Move the scale to line 3
- Move the command line to line 22
- Allow mixed case input
- Move the current line to line 4
- File the file, then **`the`** it again. Are you happy with the changes?

Introduction to Linux

Writing and Using Shell Scripts

Agenda

- Terms and concepts
- Statement types
- Invoking a shell program
- System commands
- Logic constructs
- Arithmetic and logic operators
- Functions and subroutines
- Debugging

Terms and Concepts

- BASH = “Bourne Again SHell”
- A shell script is an ordinary text file containing commands that will eventually be read by the shell
- Generally used to startup, control and/or terminate application programs and system daemons
- An interpreted language
- The first line of the program identifies the interpreter: Using **`#!/bin/<shell>`** (“shbang”) -
  - **`#!/bin/bash2`**
  - **`#!/bin/sh`**
  - If file does not have “x” privileges then: **`sh <pathname>`**

Comments

- A comment begins with the string # and ends with the end of the line
- A comment cannot span multiple lines
- It can appear on the same line as an executable statement

```
J=$(($J+1))   # Increment secondary counter
```

- It cannot be embedded in the middle of an executable statement

Simple Variables

- Symbols when first defined must begin with an alphabetic or special character "_"
  - Symbols may contain alphabetic, special, and numeric
- Symbols referred to by $<symbol name>:

```
X=1
echo $X
```

- Symbols are case-sensitive
  - **$fred** is not the same symbol as $**Fred** is not the same symbol as $**FRED**
- Symbols that have never been assigned a value have a default of ""

Assignment

- The equal sign **=** is used as the assignment operator

```
i=3
j="A string"
k_q=`expr $i + 2` or k_q=$(($i+2))
```

- It is also used as the comparison operator for numeric equality

```
if [ $i = 4 ]…
_equal =`expr $i = 4` or _equal=$(($i==4))
```

  - Usage is determined from context
    - The last statement above sets the variable **_equal** to 'true' or 'false' (1 or 0) depending on whether **$i** equals 4

Array Variables

- Arrays of values are implemented using:

```
#!/bin/bash2
Y=0
X[$Y]="Q"
echo ${X[$Y]}
-----------------------------
Q
```

Syntax

- A script may have parameters and options using the same syntax as normal commands

```
foo -anycase .therc
```

  - might perform the foo function on file **`.therc`**, ignoring case
- We must be able to perform the usual functions of a program:
  - access the parameter string
  - produce output
  - exit the program when done

Accessing Parameters

- Parameters are identified by **`$0`**, **`$1`**, **`$2`**...
- **`$0`** returns the name of the script
- **`$#`** returns number of arguments
- **`$*`** returns all arguments
- The **`set`** function can assign values to **`$0`** etc.
- The **`shift`** function makes **`$0=$1`**, **`$1=$2`** etc.

Accessing Parameters

- Use **`getopt`** function to resolve flags and operands:

```
getopt <flags> <result>
```

```
while getopts pu opt
do
   case "$opt" in
      p) _autoload_dump printable; return 0;;
      u) _autoload_unset=y ;;
      *) echo "autoload: usage: autoload [-pu] [function ...]" >&2
      return 1 ;;
   esac
done
```

The echo Instruction

- One way to produce output from a program is simply to display it on the terminal or monitor
- The **echo** instruction does this

  echo *expression*

  – evaluates the expression and displays its value
- For example

```
echo “Hello World!”
X=“XYZ”
echo $X
-----------------------------------
Hello World!
XYZ
```

Tracing the Program

- Prior to executing:

  `set -x`

- Option of **`sh`** command:

  `sh -x <shellscript>`

- Within a script:

```
#!/bin/sh
set -x
echo $0
```

Terminating the Program...

- The **exit** instruction terminates the program immediately.
- It takes an optional parameter of a return code
  - The return code must be an integer
  - It may be positive, negative, or zero

```
echo “File not found”
exit 28
```

Structure and Logic

- Several programming constructs are available in the shell language
  - The **if/then/fi** and **if/then/else/fi** constructs
    - The **else** clause is optional
    - The forms may be nested to execute complex logical operations
  - The loop constructs
    - At least five unique forms exist
    - They can be combined to produce interesting results
  - The **case … esac** construct
    - Used to execute one of a set of mutually exclusive code fragments

The Simple do…done Group

- A group of statements may be preceded by a **do** statement and followed by an **done** statement
  - This allows the group of statements to be treated as a unit
  - No change in the execution of the statements is produced
- The entire set of statements between the **do** and **done** is executed if *condition* is true

While 1 -- an Unending Loop

- The **`while 1`** or **`until 0`** construct will loop forever
- Used when the termination condition is not known
- The termination condition (if any) is found inside the group

```
while [ 1 ];
do
     ….
     if [ condition ]; then
          break
     fi
done
```

The break Instruction

- The **break** instruction is used to exit an iterative loop
- By default, it exits the innermost loop if it is executed inside nested loops then **`break`** ***n*** will exit out of *n* levels of loops
- If *n* is greater than the level of nesting then all levels are exited

Looping Through a List

- There are several forms of a **Do** loop controlled by a counter

```
for variable in list
do
    statement      Execute statement on each loop.
done        Close the do with done.
```

…Looping Conditionally

- An **until** loop always executes at least once
- A **while** loop will not execute at all if *condition* is false at initial entry to the **while** statement

```
while list do statements done

until list do statements done
```

Conditional Execution (If/Then/Else)

- Uses the traditional form of the conditional execution statements

```
if [ test ]
then
        command
else        Else is optional.
        command
fi          if always finishes with fi.
```

Tests

- The test may deal with file characteristics or numerical/string comparisons.
- Although the left bracket here appears to be part of the structure, it is actually another name for the Unix test command (located in /bin/[).
- Since [is the name of a file, there must be spaces before and after it as well as before the closing bracket.

Tests

- Examples:

```
if [ $# -ne 1 ]
then
       echo "This script needs one argument."
       exit -1
fi
input="$1"
if [ ! -f "$input" ]
then
       echo "Input file does not exist."
       exit -1
else
       echo "Running program bigmat with input $input."
       bigmat < $input
fi
```

The Case Construct…

- Many programming languages have a construct that allow you to test a series of conditions and execute an expression when a true condition is found

```
case $key in          Match the variable $key.
       pattern1)              Test match to pattern1.
          statement           If $key matches pattern1, then
                                execute statement
       ;;                     Each pattern ends with ;;.
       pattern2)              Test match to pattern2
          statement              If match, then execute
          statement
       ;;
esac           Close the case with esac.
```

The Case Construct

- The first condition that evaluates as “true” causes its corresponding expression to be executed
  - Control then transfers to the end of the `case` group
  - No other conditions are tested
- The same rules apply here for expressions as apply with the `if/then/else` construct

Arithmetic Functions…

- - + unary minus and plus
- ! ~ logical and bitwise negation
- ** exponentiation
- * / % multiplication, division, remainder
- + - addition, subtraction
- << >> left and right bitwise shifts
- <= >= < > comparison
- == != equality and inequality

Arithmetic Expressions

- & bitwise AND
- ^ bitwise exclusive OR
- | bitwise OR
- && logical AND
- || logical OR
- *expr*?*expr*:*expr* conditional evaluation
- = *= /= %= += -= <<= >>= &= ^= |= assignment

Comparison Functions

- *TEST OPTIONS - FILE TESTS*
  - `-sfile` **Test if file exists and is not empty.**
  - `-ffile` **Test if file is an ordinary file, not a directory.**
  - `-dfile` **Test if file is a directory.**
  - `-wfile` **Test if file has write permission.**
  - `-rfile` **Test if file has read permission.**
  - `-xfile` **Test if file is executable.**
  - `!` **Not operation for test.**

Comparison Functions

- *TEST OPTIONS - STRING COMPARISONS*
  - `$X -eq $Y` **$X is equal to $Y.**
  - `$X -ne $Y` **$X is not equal to $Y.**
  - `$X -gt $Y` **$X is greater than $Y.**
  - `$X -lt $Y` **$X is less than $Y.**
  - `$X -ge $Y` **$X is greater than or equal to $Y.**
  - `$X -le $Y` **$X is less than or equal to $Y.**
  - `"$A" = "$B”` **String $A is equal to string $B.**

Comparison Functions

- *TEST OPTIONS - NOT (!)*
  - `"$A" != "$B"` **String $A is not equal to string $B.**
  - `$X ! -gt $Y` **$X is not greater than $Y.**

```
#!/bin/bash2
if [ $# -ne 1 ]
then
        echo "This script needs one argument."
        exit -1
fi
input="$1"
if [ ! -f "$input" ]
then
        echo "Input file does not exist."
        exit -1
else
        echo "Running program bigmat with input $input."
        bigmat < $input
fi

exit
```

Debugging Shell Scripts

- The **set** instruction is your primary debugging tool
  - **`set -a`**
  - **`set -n`**
  - **`set -u`**
  - **`set -v`**
  - **`set -x`**