# Linux Redirection

When a user runs a Linux command the output is usually sent to the screen by default.

```
echo “Hi there”
```

displays Hi there on the screen

Output from a command can instead be sent to a file:

```
echo “Hi there” > fileX
```

Instead of displaying the message, a file called fileX is created (in the current directory) with the output of the message.

# Linux Redirection (cont.)

Redirection can use a single > or double >>.

```
echo “Hi there” > fileX
```

This creates a new file fileX – if fileX already exists, it is overwritten.

```
echo “Hi There” >> fileY
```

If fileY already exists, the output of the command is added to it. If fileY does not exist, it will be created with the output.

This can be useful for capturing data.

# Linux piping

Piping is the term given to making the output of one command the input of another. It uses the | symbol.

ps –A lists all processes

less fileX displays fileX one screen at a time

Use a pipe:

ps –A | less

Displays the output of ps –A one screen at a time.

# Linux short cuts: hard links

A **hard link** is a reference, or pointer, to physical data on a storage device. On most file systems, all named files are hard links. The name associated with the file is simply a label that refers the operating system to the actual data.

This means that more than one name can be associated with the same data. Though called by different names, any changes made will affect the actual data, regardless of how the file is called at a later time. Hard links can only refer to data that exists on the same file system. This means one cannot have hard links across different devices.

# Hard Links

- Consider the following file:

**720964** -rwxr--r-- **1** bill staff 49 Feb 6 13:22 backup

- The 720964 value is the i-node and the **1** indicates that this file is linked to data. Issue the command: **ln backup backup2**

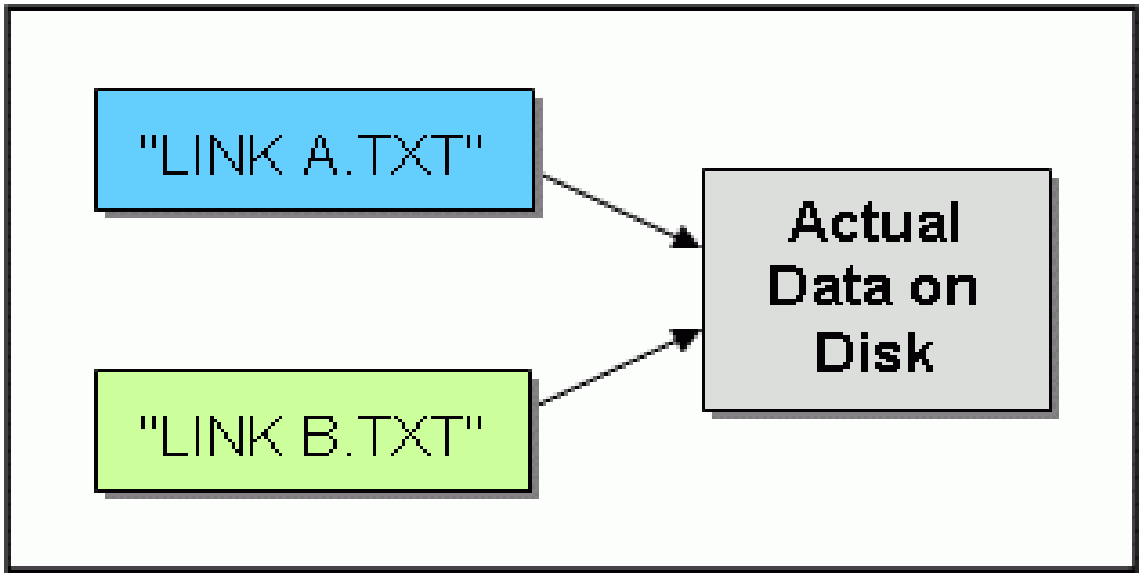
- Now

**720964** -rwxr--r-- **2** bill staff 49 Feb 6 13:22 backup
**720964** -rwxr--r-- **2** bill staff 49 Feb 6 13:22 backup2

This looks like two files but only one exists with two links to it – one i-node number.

# Hard Links

The two hard links named "**LINK A.TXT**" and "**LINK B.TXT**" have both been linked i.e. made to point to the same physical data.



# Linux short cuts: symbolic (soft) links

- A **symbolic link** (often known as a ***soft link***) consists of a special type of file that serves as a reference to another file.

- Unlike a hard link, a symbolic link does not point directly to data, but merely contains a symbolic path which an operating system uses to identify a hard link (or another symbolic link).

# Soft Links (cont.)

- Consider again our file:

**720964 -rwxr--r-- 1 bill staff 49 Feb 6 13:22 backup**

- Issue the command: **ln -s backup backup3**

**720964 -rwxr--r-- 1 bill staff 49 Feb 6 13:22 backup**

**720971 lrwxrwxrwx 1 bill staff 6 Feb 13 17:26 backup3 -> backup**

**A new file has ben created.**

# Soft Links (cont.)

- Soft links are useful in making connections across different partitions including different media (e.g. USB devices).
- Soft links can be found in run level files.
- In Ubuntu, look in the directory: /etc/rc5.d

# Soft Links (cont.)

From Linux (/etc/rc5.d):

```
lrwxrwxrwx 1 root root 10 Mar  2  2011 K01cpufreq -> ../cpufreq
lrwxrwxrwx 1 root root  7 Sep 17  2012 K01cron -> ../cron
lrwxrwxrwx 1 root root 16 Mar  2  2011 K01microcode.ctl -> ../microcode.ctl
lrwxrwxrwx 1 root root  7 Mar  2  2011 K01nscd -> ../nscd
lrwxrwxrwx 1 root root  8 Sep 17  2012 K01pcscd -> ../pcscd
lrwxrwxrwx 1 root root  9 Mar  2  2011 K01random -> ../random

lrwxrwxrwx 1 root root 10 Mar  2  2011 S01cpufreq -> ../cpufreq
lrwxrwxrwx 1 root root  7 Mar  2  2011 S01dbus -> ../dbus
lrwxrwxrwx 1 root root 14 Mar  2  2011 S01earlysyslog -> ../earlysyslog
lrwxrwxrwx 1 root root  8 Mar  2  2011 S01fbset -> ../fbset
lrwxrwxrwx 1 root root 16 Mar  2  2011 S01microcode.ctl -> ../microcode.ctl
```

# Environmental Variables

- The environmental variable describes a mechanism for defining variables that can be used to hold pieces of information for use by the system programs or for the user's own use. They can be very useful when writing shell scripts.

- The command ***env*** *lists all Linux environmental variables and their values*

# Environmental Variables (cont.)

- The current value of each variable is displayed using the echo command with the $ symbol e.g. echo $PATH

- Examples: PATH, HOME, PS1, LOGNAME & SHELL

# PATH

**PATH**: This is a list of directories that the shell searches when we enter a command. For example, to run a command such as ls – the ls command has to be in a directory that's in the PATH.

**echo $PATH**

/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:
/usr/X11R6/bin:/bin:/usr/games:/opt/gnome/bin:/opt/kde3
/bin:/usr/lib/jvm/jre/bin

Note: the ls is usually in /bin or /usr/bin

# PATH (windows)

PATH is used in the windows in the same way it's used in Linux.

C:path

PATH=C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;
C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32
\WindowsPowerShell\v1.0\;C:\Program Files\Intel\Intel(R) Management Engine Compone
nts\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Prog
ram Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files
(x86)\Intel\Intel(R) Management Engine Components\IPT

Similar appearance and identical purpose as PATH in Linux

# PS1, PS2 & HOME

- **PS1** and **PS2**: The primary and secondary shell prompts respectively.

  echo $PS1

  \h:\w #linux:~ #

- See http://www.cyberciti.biz/tips/howto-linux-unix-bash-shell-setup-prompt.html

- **HOME**: This is the pathname to your home directory – set when you log on.

  echo $HOME

  /home/cadmin

# SHELL & LOGNAME

- **SHELL**: This is the pathname of the shell you are using.

  echo $SHELL

  /bin/bash

**LOGNAME**: Your username.

  echo $LOGNAME

  cadmin

# Combine 2 Environmental Variables

It’s possible to customise a prompt.

PS1="Hello $LOGNAME Welcome to Linux >"

Hello root Welcome to Linux >

**or**

PS1=”$LOGNAME’s Linux >”

cadmin’s Linux >

# Introduction to Scripting

Script

A script is a text file that contains Linux commands.

BASH Script – a script written in the bash syntax.

A script is executed i.e. it’s an operating systems program.

Scripts are useful in Linux administration

# Script Requirements

- An editor is required e.g. nano
- Decide where the script is to be stored
- Write the script
- Grant permission to execute
- Execute the script

# Script Example

The command echo “Hello Everybody” displays the message Hello Everybody on the terminal screen.

This command can be saved in a file and run as a script.

Open an editor, enter the text: echo “Hello Everybody”

save as: myFirstShell

Grant permission to execute:

chmod u+x myFirstShell

Execute by entering: ./myFirstShell

# Script Example No. 2

The following example lists a number of commands. Open the editor, enter text

```
clear
date
echo
cal
echo
ls –l
```

Save as script2.

Grant execute permission: chmod u+x script2

Execute: ./script2

# Starting Linux (booting)

The kernel has to be loaded into memory to a point where the first process starts.

This involves using a special program called a boot loader, mounting the file system (i.e. making it available), starting the first process (e.g. init) and finally enabling a user to log on.

# Steps in booting Linux

The BIOS loads and runs a boot loader

The loader locates the kernel on disk, loads it into memory and starts it.

The kernel initialises devices.

The kernel mounts the root filesystem

The kernel starts the first process

This starts other system processes

Eventually a process is started which allows user to log in

# The boot loader

There are a number of boot loaders available such as:

GRUB – usually standard

LILO – an original Linux loader

# Dual boot

Hard disks can be divided into four primary partitions.

It's possible for each partition to have an operating system.

When the boot loader runs, it offers the user a menu as to which operating system to load.

Only one operating system can be used at a time

If additional partitions are required, a primary partition can be marked as extended and then divided into a number of logical partitions.

# Booting from other devices

Operating systems can be booted from USB storage devices and CD / DVD media.

Interrupt the startup and enter the BIOS setup

Change the boot default to boot from selected device.

# Loading Linux from a virtual machine

Having an operating system on a virtual machine (like vmWare or vmPlayer) means a user can access both operating systems.

There is an overhead to using a virtual machine (e.g. might have poor performance if dealing with 3 D graphics when playing a game).

# Linux Protection

Protection and security are very important in any system.

Two features of basic protection:

- Protecting a system from unauthorised access
- Managing file permissions for legitimate users

# Linux Protection: passwords

Linux stores the encryptions of users' passwords in /etc/shadow which cannot be read by ordinary users.

Linux uses one-way encryption. A good one way encryption algorithm means that:

- It is not possible to derive the original password from the encrypted string
- Each encryption string has the same number of characters, regardless of the length of the password.
- No two passwords should result in the same string.

# Examples of shadow file records

cadmin:$6$w4JRvxvM$r2xRS/KH2cgX0SmwXS4UA1GZry.0RrWstfgYgEfTEbAqMQnxoLF0dLXNQ8NAgYr2ZYxQyAZr.60RUC0HljZLm0:17424:0:99999:7:::

user1:$6$FBfJeOumfn2B/zuf$OMYgJ1IhM6jnPY8Lf5bqm1It.KTZFNiskRYc4QHqF94U6KikJ47lhKCfKICYnmuV1eKzXMWp3Bk7Q4pPD7/Uq/:17567:0:99999:7:::

user2:$6$UzMBPQkIsOeSff$Uwr.BmYwhyl7mMTitfLv40fTc.L9MYbTZhu8wfhJGcNlDf.OFrtZdywVBwjQHZlfXmNun1Ua4TCYryd17YDDW1:17567:0:99999:7:::

user3:$6$XYNncDgSlGJ9MM5p$ivaW9cl3deRvF7PWEnWuKzRh2NL/9bpVjXtAO1Nnjx45TuWfcyYer8dTVx1FemJI/jlFqti/ipDd4PD8Q1A1o.:17567:0:99999:7::

Note:

Each string has the same number of characters

Two passwords are the same but have different encrypted outputs (because of salt)

# Linux salt

For additional protection, Linux adds an additional (random) character to each password before encryption.

This means that even if different users use the same password – the encrypted strings would be different.

It also makes a ‘dictionary’ attack more difficult – where a hacker tries to access a system with a list of potential passwords.

# Good password guide

The first line of defence is the password.

Passwords should be:

- secret
- not a dictionary word
- consist of upper and lower characters
- at least 7 / 8 characters long
- not the name of a person or place
- not a combination of a name and number e.g. spring22
- combination of alphanumeric characters
- changed regularly

# Cracking a password

Dictionary Attack: a hacker uses software to ‘guess’ a user’s password against a dictionary of common passwords including names of people and places.

Brute Force attack: tries every possible combination of letters and characters in an effort to crack a password. If it’s a good password this should take several weeks.

# Cracking a password

There are 95 printable characters available for use as part of a password.

The more characters in a password - the more permutations and the more difficult it is to crack.

For example:

1 character password - 95 permutations

2 character password – $95^2$ (= 9025) permutations

8 character password – $95^8$ (= 6,634,204,312,890,625) permutations

Each additional character exponentially increases the number of permutations.

# File Permissions

File permissions provide the primary means of internal security.

Each file has nine permissions. These are broken down into groups of three:

owner :read, write + execute

group: read, write + execute

others: read, write + execute

Only the owner of a file and root can change permissions

# Regular File Permissions

**Recap from Week 3**

Example

-rwxr-xr-- 1 joe groupX 34 Feb 1 12:50 myFile

This is an ordinary file

The owner of the file is joe

joe can read (view file contents), write (make & save changes) and execute myFile (if it's a program or a script)

Members of groupX can read and execute myFile

Other system users can read myFile

# Directory Permissions

**Recap from Week 3**

Directories also have permissions but have slightly different meanings.

Example

```
drwxr-x--  joe   groupY  1024  2 Feb 12:50  myDir
```

joe owns this directory and can change any of its nine permissions

joe can see directory contents and add or remove files to / from it.

Members of groupY can see directory contents (& copy them) but can not add or remove contents.

Other users have no permission.

# Changing File Permissions

- To add permission:

  **chmod u + x temp** (grants the owner executable rights)

- To remove permission:

  **chmod g – r temp** (removes read from the group rights)

- More than one permission can be changed at a time, for example:

  **chmod u + rx** (grants user read and executable rights)

# Changing permissions (cont.)

Permissions can be changed by the owner of a file using the chmod command.

Permissions are added using +

Permissions are removed using –

For the file: -rwxr-xr-- 1 joe groupX 34 Feb 1 12:50 myFile

chmod u - w myFile
chmod g + w myFile
chmod o – r myFile

results in

-r-xrwx--- 1 joe groupX 34 Feb 1 12:50 myFile

# Changing permissions (octal notation)

Using octal notation, all nine file permissions can be changed using one command.

A three digit number changes every permission.

Each digit has a value between 0 and 8.

The first digit is for all owner permissions.

The second for group permissions

The third for other permissions

# Octal Notation

To understand the use of octal notation, think of each of the three sets as being a separate number between 0 and 7. Notice how a 1 in the *binary* column means that there will be a corresponding permission in the *permissions* column

| Octal | Binary | Permission | English Translation |
|---|---|---|---|
| 0 | 000 | --- | No permissions. |
| 1 | 001 | --x | Execute only. |
| 2 | 010 | -w- | Write only. |
| 3 | 011 | -wx | Write and execute. |
| 4 | 100 | r-- | Read only. |
| 5 | 101 | r-x | Read and execute. |
| 6 | 110 | rw- | Read and write. |
| 7 | 111 | rwx | Read, write, and execute. |

An octal value can be used to set all permissions. Set the file permissions for **temp** to **rwxr-x---**

**chmod 750 temp**

7 (rwx), 5 (r-x) & 0 (---).

# Changing file owner & group

Only the root user is able to change the owner or the group of a file

The Linux commands for changing file owner and group are:

**chown & chgrp**

Examples:

sudo chown dave fileX makes dave owner of fileX

sudo chgrp groupX fileX changes group of fileX to groupX

# Creating a group & adding members

Only the root user can create a group and add users to it.

This can be done using a GUI or alternatively from the terminal:

```
sudo groupadd newGroup
```

Check the group file (cat /etc/group) to see that the group now exists

To add members to the group:

```
usermod –a –G newGroup joe
```

Note: joe has to first have an account.

# Controlling Access

- File permissions, directory permissions, file ownership and directory ownership allow users to control access to their own accounts. The root user can also use permissions to control access to system resources.

- Consider the file properties e.g. **ls –l**

  **-rw-r--r--1 bill users 10 Oct 15:15 myFile** (ordinary file)

  **drwxr-xr-x 2 bill users 10 Oct 15:15 bill** (directory)

# Controlling Access –Case 1

- There are three users **bill**, **kate** and **jane**. We want to allow **bill** and **kate** access to a browser; **kate** and **jane** access to a compiler.

- For the purpose of this exercise we will assume the access to these resources is controlled by two files **browserFile** and **compilerFile**. We will also assume that they in a directory all users can access (e.g. **/media**).

# Case 1 (cont.)

Create two groups: browserGroup & compilerGroup

sudo groupadd browserGroup

sudo groupadd CompilerGroup

Check the group file to see these are created.

# Case 1 (cont.)

- Put **bill** and **kate** in browser group – **browserGroup**
- Put **kate** and **jane** in compiler group – **compilerGroup**

```
usermod –a –G browserGroup bill

usermod –a –G browserGroup kate

usermod –a –G compilerGroup kate

usermod –a –G compilerGroup bill
```

# Case 1 (cont.)

- If we go to the directory and do a long listing:

  **cd /media**
  **ls -l**

- We might see something like:

  **-rw-r—r--1 root root 8 Oct 15:15 browserFile**

  **-rw-r--r--1 root root 8 Oct 15:15 compilerFile**

# Case 1 (cont.)

- Change the group of each of these to the appropriate group as follows:

  **sudo chgrp browserGroup browserFile**

  **sudo chgrp compilerGroup compilerFile**

- If we do another long listing (**ls –l**):

  **-rw-r--r-- 1 root browserGroup 8 Oct 15:15 browserFile**

  **-rw-r--r-- 1 root compilerGroup 8 Oct 15:15 compilerFile**

# Case 1 (cont.)

- Change permissions:

**sudo chmod g + x browserFile**

**sudo chmod o – r browserFile**

**sudo chmod g + x compilerFile**

**sudo chmod o – r compilerFile**

# Case 1 (cont.)

- Now if we do a long listing (**ls –l**)

  **-rw-r-x---1 root browserGroup 8 Oct 15:15 browserFile**

  **-rw-r-x--- 1 root compilerGroup 8 Oct 15:15 compilerFile**

- Now (apart from root) only members of **browserGroup** and **compilerGroup** can run the browser and compiler respectively**.**

# Case 2

- There are three users **bill**, **kate** and **jane**. We want to allow **jane** and **kate** to be able to copy files from each other's home directories and to prevent **bill** from doing it.

- In order to copy files from each other, users require access to each others home directories i.e. require read and execute permission. Assume home directories are sub-directories of **/home** i.e.

# Case 2 (cont.)

**cd /home**

**ls –l**

**drwxr-xr-x 2 kate kate 8 Oct 14:10 kate**

**drwxr-xr-x 2 jane jane 8 Oct 14:10 jane**

**drwxr-xr-x 2 bill bill 8 Oct 14:10 bill**

# Case 2 (cont.)

Create a new group e.g. **groupX**

sudo groupadd groupX

Add **jane** and **kate** to **GroupX**.

usermod –a –G groupX kate

usermod –a –G groupX jane

# Case 2 (cont.)

Change the group owner of **kate**'**s** and **jane**'**s** directories.

Directories are treated the same as files in terms of their properties and the commands that change these properties.

**sudo chgrp groupX kate**

**sudo chgrp groupX jane**

**Note:** **kate & jane here refer to the home directories of jane & kate**

# Case 2 (cont.)

**ls -l**

**drwxr-xr-x 2 kate groupX 8 Oct 14:10 kate**

**drwxr-xr-x 2 jane groupX 8 Oct 14:10 jane**

**drwxr-xr-x 2 bill bill 8 Oct 14:10 bill**

# Case 2 (cont.)

Change the permissions:

**sudo chmod o-rx kate**

**sudo chmod o-rx jane**

Do a long listing (**ls –l**):

**drwxr-x--- 2 kate groupX 8 Oct 14:10 kate**

**drwxr-x--- 2 jane groupX 8 Oct 14:10 jane**

**drwxr-xr-x 2 bill users 8 Oct 14:10 bill**

# Case 2 (cont.)

With these permissions:

**drwxr-x--- 2 kate groupX  8 Oct 14:10 kate**

**drwxr-x--- 2 jane groupX  8 Oct 14:10 jane**

**drwxr-xr-x 2 bill users  8 Oct 14:10 bill**

kate & jane can access each other's home directories as each belongs to groupX.

Other users (not in groupX) have no permissions.