Linux Training Materials Project

GBdirect Limited
27 Park Drive
Bradford, BD9 4DS
West Yorkshire
tel: +44 (0)1274 772277
linux@gbdirect.co.uk

November 7, 2001

Contents

Module 1

Apache Basics

Objectives

On completion of this module you should be able to:

- Install and configure the `Apache` webserver
- Set up *virtual hosts*
- Use access controls
- Set up basic authentication

1.1 What is `Apache`?

- `Apache` is the most widely-used web-server[1]
- Listens for requests and hands something back
- Normally the contents of a file
  - ♢ Possibly the result of a program
- Designed to be stable and configurable
  - ♢ Not meant to be fast
  - ♢ Probably fast enough . . .

[1]59.99 % of all servers as of February 2001 (Netcraft - www.netcraft.com)

1.2 Installation

- Basic installation is easy
- You may be able to install from your distribution
  - ♢ Most come with `Apache`
- Otherwise just follow the download instructions from the official site
  - ♢ `http://www.apache.org/`
- Then follow the instructions in the `INSTALL` file
  - ♢ Normally just

    ```
    $ ./configure
    $ make
    $ make install
    ```

  - ♢ If you have problems check the docs
  - ♢ Available at `http://www.apache.org/docs`

1.3 How Apache Listens

- Apache runs several processes at any one time
  - ♢ Parent and several children
- Parent *‘watches over’* the children
  - ♢ Tracks how many are answering requests
  - ♢ Spawns more if free processes drop below a certain point
  - ♢ Kills spare processes if there are lots free
- Configure child numbers using *MinSpareServers* and *MaxSpareServers* directives
  - ♢ Default is reasonable for a small business
  - ♢ Tune it for busier sites

1.4 Configuration File(s)

- If compiled from source, `Apache` installs in `/usr/local/apache`
  - ◊ Earlier versions installed under `/usr/local/etc/httpd`
  - ◊ Your distribution may differ again . . . [2]
- Configuration file is called `httpd.conf`
  - ◊ Older versions use [3]
    - ∗ `httpd.conf`
    - ∗ `srm.conf`
    - ∗ `access.conf`
- Controls what requests `Apache` answers
  - ◊ and how . . .

[2]Redhat installs config files under `/etc/httpd` and the sample web pages and logs directories under `/home/httpd`

[3]Some pre-packaged versions (Such as the RedHat RPM) also use separate files

1.5 Key Configuration Directives

- Wide range of *configuration directives*
- For a *very* basic server you need at least the following:
  - ♢ *ServerRoot*
  - ♢ *DocumentRoot*
  - ♢ *ServerAdmin*
  - ♢ *BindAddress*
  - ♢ *Port*
  - ♢ *Listen*
  - ♢ *User*
  - ♢ *Group*

1.6 *ServerRoot*, *DocumentRoot*

- Tells `Apache` where its files live
- *ServerRoot* tells `Apache` where its `conf` and `logs` directories live
  - ◇ Not always necessary
  - ◇ Good practice to have it
- *DocumentRoot* tells `Apache` where to look for documents to serve up
- Requested filenames are appended to this
- If you have

  ```
  DocumentRoot /www/docs
  ```

  then a request to

  ```
  http://www.domain.co.uk/foo.html
  ```

  points to the file `/www/docs/foo.html`

1.7 *ServerAdmin*

- `Apache` sometimes can’t complete requests
- In these cases it serves up an error page
- *ServerAdmin* is given as a contact address
- Usually set to something like

  `webmaster@domain.co.uk`

  ◊ You should of course ensure that it is a *valid* email address
- Possible to specify a different error page

  ◊ Doesn’t have to use ServerAdmin

1.8 *BindAddress*, and *Port*

- Tells `Apache` which requests to answer
- By default `Apache` listens to every IP address on your machine
  - ◇ But only to the port given by the *Port* directive
- `BindAddress 192.168.0.1` tells `Apache` to ignore anything that doesn't come in on `192.168.0.1`
- `Port 8080` ignores all but the specified port
- You can use more than one *Port* directive, e.g.

```
Port 80
Port 8080
```

- If you don't specify a port then a default is used[4]
- You can only use one *BindAddress*!

[4]This is usually 80, but if you are using a binary package then bear in mind whoever compiled your package may have chosen a different value

1.9 *Listen*

- *Listen* is a replacement for *BindAddress* and *Port*
- Given IP:port or just port, e.g.

  ```
  Listen 192.168.0.1:8080
  ```

  will answer requests on the IP address `192.168.0.1` and port `8080` and no others
- To answer requests to all valid IP addresses, but only a certain port (e.g. `80`) use:

  ```
  Listen 80
  ```
- Can use more than one *Listen* directive
- Should be used instead of *BindAddress* and *Port* in new servers

1.10 *User* and *Group*

- `Apache` should normally be started as *root*
  - ♢ So it can change the user ID of the children
  - ♢ These should *not* run as root
- *User* and *Group* directives say what user/group the children should run as
  - ♢ Important security feature
- Should be set to something that has no real power on your system
  - ♢ Most people use user and group `nobody`
- Web documents should be readable by this user
- Nothing should be writeable except log files

1.11 Apache Processes

- Looking at a process list[5] you can see
  - ♢ The parent

    ```
    root S Jul  4  0:37 /usr/local/apache/bin/httpd -d /www
    ```
  - ♢ The children

    ```
    nobody  S  10:11  0:00 /usr/local/apache/bin/httpd -d /www
    nobody  S  10:20  0:00 /usr/local/apache/bin/httpd -d /www
    nobody  S  10:58  0:00 /usr/local/apache/bin/httpd -d /www
    nobody  S  10:58  0:00 /usr/local/apache/bin/httpd -d /www
    nobody  S  11:06  0:00 /usr/local/apache/bin/httpd -d /www
    nobody  S  11:13  0:00 /usr/local/apache/bin/httpd -d /www
    ```
- Spare processes don't use processor time
  - ♢ They are *'sleeping'*
- They *do* use memory, however
  - ♢ Negligible for a default `Apache`
  - ♢ Watch carefully the more modules you add!

[5]Some fields from the `ps` output have been left out to aid clarity

1.12 Logging

- `Apache` can log information about accesses
- Use the *TransferLog* and *ErrorLog* directives
- `TransferLog logs/access_log`
  will log all requests in the file
  `ServerRoot/logs/access_log`
- If the filename starts with a `/` then it is treated as a proper pathname, not appended to *ServerRoot*
- *ErrorLog* is similar but controls where error messages go
  - ♢ Useful for debugging CGI scripts and misconfigurations
  - ♢ Check here first if `Apache` won't start

1.13 Customizable Logging

- Customizable logs available with *CustomLog*

  `CustomLog filename format-string`

- `format-string` consists of *‘% directives’* and/or text
- *% directives* include:

`%b`	Bytes sent, excluding HTTP headers
`%f`	Filename
`%{headername}i`	The contents of headername: header in the request
`%P`	The process ID of the child that serviced the request
`%r`	First line of request
`%t`	Time, in common log format time format
`%T`	The time taken to serve the request, in seconds
`%u`	Remote username (may be bogus if return status (%s) is 401)
`%U`	The URL path requested
`%v`	The ServerName of the server answering the request

1.14 *CustomLog* examples

- To log the referer information in the file `ServerRoot/logs/referer`

```
CustomLog logs/referer "%r Refered by: %{Referer}i"
```

- *% directives* can be conditional on reply status

```
CustomLog logs/referer "%r Refered by: %200,304,302{Referer}i"
```

  - ◇ Logs the refering page only on status 200,304,302 [6]

- For full details consult the Apache documentation
  - ◇ Gives list of all possible *% directives*

[6]For full details consult the Apache documentation

1.15 Example Configuration

- A sample configuration file could look like this :

```
ServerRoot /usr/local/apache
DocumentRoot /usr/local/apache/htdocs
ServerAdmin webmaster@domain.co.uk
Listen 192.168.0.131:80
User nobody
Group nobody
ErrorLog /usr/local/apache/logs/error_log
```

- We recommend starting with the default `httpd.conf` rather than from scratch
  - ♢ Correctly configures many things for you
- The default is well annotated
  - ♢ Everything after a # character is a comment
  - ♢ Ignored by `Apache`
- `Apache` can check the syntax of its configuration
  - ♢ `httpd -t`
  - ♢ `apachectl configtest`

1.16 Basic Exercises

1. *Apache Installation*
   (a) Find out if Apache is installed on your machine . . . if not, install it.
   (b) Check Apache is running on your system.
      i. You should be able to point your web browser at `http://127.0.0.1/` to check this
      ii. You might have to try `http://127.0.0.1:8080/`
   (c) If Apache is not running, start it
      i. Run `/usr/local/apache/bin/apachectl --help` for information
   (d) If Apache still doesn't appear to be running, find its configuration and log files and try to fix the error.
2. *Basic configuration*
   (a) Familiarise yourself with the `httpd.conf` file.
   (b) How would you change the directory where the log files are kept?.
   (c) How would you change the 'root' for documents?
   (d) How would you enable symbolic links to be followed on the cgi-bin directory.
   (e) Make your site only accessible on Port 8080
   (f) Now make it only accessible on the IP address 127.0.0.1, and port 80
   (g) Make the changes and check them.
   (h) Place the following line in your `/etc/hosts` file:

   ```
   IP_ADDRESS www.test.co.uk www
   ```

   where `IP_ADDRESS` is the IP address of your machine. You should now be able to browse `http://www.test.co.uk/`
3. *Logging*
   (a) Take a look at the access logs and familiarise yourself with the information they contain.
   (b) Set up a custom log to give the time of the request, the request, referer, and number of bytes sent, as well as the time taken to serve the request.
   (c) Alter your custom log to show the time taken and bytes sent *only* if a 200 status response occured.

1.17 Solutions

1. *Apache Installation*

   (a) If Apache is not installed you should be able to install it off a RedHat CD by mounting the CD and typing `rpm -ivh /mnt/cdrom/RedHat/RPMS/apache.rpm`

   (b) There are several ways to check this. One is to `telnet` to port 80 of your machine and see if you get a response.

      i. This should work for a default RedHat install, though the port number that Apache first listens on changes in various different packaging so you should try both 80 and 8080.

   (c) You can start Apache one of two ways (Which may be the same on some machines!)

      - /etc/rc.d/init.d/httpd start
      - somepath/apachectl start [7]

   (d) If you can't work out why Apache isn't running ask the tutor for assistance.

2. *Basic configuration*

   (a) You should make sure that you understand everything in the `httpd.conf` including those sections that are commented out.

   (b) Alter the `CustomLog` and `ErrorLog` directives to change where the log files are kept, e.g.

```
ErrorLog /var/log/myerrorlog
CustomLog /var/log/myaccesslog common
```

   (c) The 'root' for documents is specified by the `DocumentRoot` directive, e.g.

```
DocumentRoot /path/to/my/web/documents
```

   (d) You can enable symbolic links by adding `Options +ExecCGI` to the `<Directory>` section for your `cgi-bin`, e.g.

```
<Directory /path/to/my/web/documents/cgi-bin>
Options +ExecCGI
</Directory>
```

   (e) Add/Change the `Port` directive in your `httpd.conf` file to read `Port 8080`

   (f) Add the following to your `httpd.conf`: `Listen 127.0.0.1:80`

   (g) Restart the server and try to access it on both port 80 and 8080. Check that it only works as you expect and fetches documents from the correct place.

   (h) Check that you can browse `http://www.test.co.uk`

[7] You may have to dig a little to find where this script is

3. *Logging*

   (a) Make sure you understand what each of the columns in the access logs is for. Try `tail`ing the logs as you browse your webserver

   (b) The following should create a file `newlogformat` which holds the desired log format.

   ```
   LogFormat "%t %U %{Referer}i %b %T" newlog
   CustomLog logs/newlogformat newlog
   ```

   (c) Change your LogFormat line to

   ```
   LogFormat "$t %U %{Referer}i %200b %200T" newlog
   CustomLog logs/newlogformat newlog
   ```

1.18 Two sites and more ...

- `Apache` can serve multiple sites easily
- Known as *'Virtual Hosting'*, e.g.

  ```
  <VirtualHost 192.168.0.2>
  DocumentRoot /www/web.test2/docs
  ServerName www.test2.co.uk
  ServerAdmin www@test2.co.uk
  ErrorLog /www/web.test2/logs/error_log
  TransferLog /www/web.test2/logs/access_log
  </VirtualHost>
  ```

  to make apache answer requests to address `192.168.0.2` from `/www/web.test2/docs`
- Your machine must answer to this address[8]
- `Apache` must be listening on the address
  - ♢ `Listen 80` will make `Apache` answer to all available addresses on port 80
- This is known as IP-based virtual hosting

[8] If you don't know how to set up IP aliases ask the instructor

1.19 Virtual Hosting Options

- IP-based
  - ♢ Each site must have a unique, IP address
  - ♢ Uses up valuable IP addresses
  - ♢ Site accessible by all browsers
- Name-based
  - ♢ Sites share an IP address
  - ♢ Useful if short of available addresses
  - ♢ Some browsers may have problems
- Most use IP-based hosting where possible
- Ensures maximum accessibility

1.20 Name-based hosting

- Name-based hosting looks like:

```
NameVirtualHost 192.168.0.3

<VirtualHost foo.domain.co.uk>
ServerAdmin foomaster@domain.co.uk
DocumentRoot /www/foo/docs
ErrorLog /www/foo/logs/error_log
TransferLog /www/foo/logs/access_log
</VirtualHost>

<VirtualHost bar.domain.co.uk>
ServerAdmin barmaster@domain.co.uk
DocumentRoot /www/bar/docs
ErrorLog /www/bar/logs/error_log
TransferLog /www/bar/logs/access_log
</VirtualHost>
```

1.21 Name-based hosting (continued)

- `NameVirtualHost` tells `Apache` that an IP address can serve multiple hosts
- `VirtualHost` sections describe how documents for each site are served
  - ◇ `Apache` must be able to resolve the names in the <*VirtualHost*> directives to the IP address
- `Apache` looks at the `Host:` header to decide which documents to serve
  - ◇ Not sent by all browsers
- Requests on other IP addresses will be processed as normal
- Can use both IP-based and name-based hosting

1.22 Block Directives

- `Apache` has several *block directives*
  - ◊ Limit enclosed directives to apply to a certain set of *'things'*
- `<VirtualHost>` is a block directive
  - ◊ Enclosed directives apply only to that virtual host
- Others are:

```
<Directory> ... </Directory>
<DirectoryMatch> ...
</DirectoryMatch>
<Files> ... </Files>
<FilesMatch> ... </FilesMatch>
<Location> ... </Location>
<LocationMatch> ... </LocationMatch>
```

1.23 Block Directives (continued)

- `<Directory name>` Limits the enclosed directives to apply to everything below the directory `name`
  - ♢ `name` can be anywhere on the filesystem
  - ♢ Independent of *DocumentRoot*
- `<Location name>` is similar but is a URL path rather than a filesystem path
- `<Files name>` limits directives to files called `name`
  - ♢ Path of the file is irrelevant
  - ♢ Only checks the file name, not its location

1.24 *DirectoryMatch*, et al.

- `DirectoryMatch`, `FilesMatch` and `LocationMatch` are similar
  - ◇ Accept regular expressions as arguments, e.g.

    ```
    <FilesMatch .*\.cgi>
    ...
    </FilesMatch>
    ```
- More flexible
- Need more thought to match *only* intended files

1.25 Access Control

- Create a file in the directory to be protected
  - ♢ Usually `.htaccess` or `.acl`
  - ♢ Can be anything
- Example:

```
AuthType        Basic
AuthName        "Members Only"
AuthUserFile    /www/conf/auth/auth.user
AuthGroupFile   /www/conf/auth/auth.group
require group   testgroup
require user    testuser
```

- Only the user `testuser`, or a user in the group `testgroup`, may access files in this directory
- Validation is done on the files
  `/www/conf/auth/auth.user`
  and
  `/www/conf/auth/auth.group`

1.26 Access Control (continued)

- Access control is *off* by default
  - ◇ Unnecessary for most sites
- Switched on by:

```
AccessFileName .acl

<Directory /www/web.test2>
AllowOverride AuthConfig
</Directory>
```

- `AccessFileName` identifies which filename(s) constitute an Access Control File
- Every directory in the request path is checked for a relevant file
- `AllowOverride` says that Access Control files can override authorisation directives only
  - ◇ Can have other values
  - ◇ Change behaviours through your `.acl` file
  - ◇ See `Apache` docs for further details

1.27 Authorisation Files

- Authorisation files are very straightforward
- Group file is `groupname: userlist`
- For example:

```
firstgroup: user1 user2 user3
secondgroup: user2 user3 user4
othergroup: user4 user5 user6
```

- Listed users belong to that group
- Must create this file by hand

1.28 Authorisation Files (continued)

- User file is a little more complicated
- Format is `username:encryptedpassword`
- For example:

```
testuser:6SlrYaxUFml
```

- Create/edit this with `htpasswd`
  - ◊ Part of the `Apache` distribution
  - ◊ Give it an authorisation file and a username

```
$ htpasswd /www/conf/auth/auth.user newuser
New password:
Re-type new password:
Adding password for user newuser
```

1.29 Other useful directives

- There are around 200 Apache directives
  - ◇ More if you add modules e.g. `mod_ssl`
- The previous ones are the *essentials*
- Some other useful directives are given below:

Directive	Action
Redirect `url-path new-url`	Redirect Requests to `url-path` to `new-url`
RewriteRule `pattern new-pattern`	Rewrite requests, replace `pattern` with `new-pattern`
AddEncoding type ext	Serve up documents with extension ext with encoding type type
ForceType type	Force all documents to be served up with MIME type type
HostNameLookups on|off|double	Whether to do DNS lookups for logging purposes
ExpiresDefault	Set the default expiry time of documents

1.30 Examples

```
Redirect permanent /ents/theatre/fab-gere http://www.fabgere.com
Redirect /gbdirect/logo.gif http://www.gbdirect.co.uk/logo.gif
Redirect permanent /gbdirect http://www.gbdirect.co.uk/

RewriteEngine on
RewriteRule ^/linuxtraining.*\.htm /ltcu_moved.htm

<Location /LTCU>
AddEncoding x-gzip gz
</Location>

<Location /LTCU-plain>
ForceType text/plain
</Location>

HostNameLookups off

<Location /LTCU>
ExpiresDefault "access plus 1 month"
ExpiresByType text/html "access plus 1 week"
</Location>
```

1.31 Exercises

1. *IP based hosting*
   (a) Start with the default installation file and add an IP based virtual host:
      i. Add an IP alias for your machine (Ensure it doesn't clash with any others on your network!)
      ii. Create a dummy index page so you will be able to tell the difference between your two sites.
      iii. Set up Apache to serve this site and check from a browser that everything works (for both sites) as you expected.
2. *Name based hosting*
   (a) Set up your apache so that it will serve the same sites but on a single IP address (Name-based virtual hosting).
3. *Access control*
   (a) Create two directories on one of your sites and set up access controls so that anyone can see the main index page, testuser can see the first directory and anyone in group testgroup can see the second.

1.32 Solutions

1. *IP based hosting*

   (a) The first thing that you will have to do is set up an IP alias for your machine so that it has two distinct IP addresses. Yu might find it easiest to use the Redhat control-panel for this. If you aren't sure how to achieve this ask the instructor. A list of spare IP addresses will be made available. An example from a working multi-hosted Apache is given below

```
Listen 192.168.0.3:80
Listen 192.168.0.2:80

<VirtualHost 192.168.0.3>
ServerAdmin webmaster@gbdirect.co.uk
DocumentRoot /home/www/web.llord/docs
ServerName llord.gbdirect.co.uk
ErrorLog /home/www/web.llord/logs/error-log
TransferLog /home/www/web.llord/logs/access-log
</VirtualHost>

<VirtualHost 192.168.0.2>
ServerAdmin webmaster@gbdirect.co.uk
DocumentRoot /home/www/web.trainingpages/docs
ServerName trainingpages.gbdirect.co.uk
ErrorLog /home/www/web.trainingpages/logs/error-log
TransferLog /home/www/web.trainingpages/logs/access-log
</VirtualHost>
```

2. *Name based hosting*

   (a) An equivalent example using name-based hosting would be:

   ```
   NameVirtualHost 192.168.0.2

   <VirtualHost llord.gbdirect.co.uk>
   ServerAdmin webmaster@gbdirect.co.uk
   DocumentRoot /home/www/web.llord/docs
   ServerName llord.gbdirect.co.uk
   ErrorLog /home/www/web.llord/logs/error-log
   TransferLog /home/www/web.llord/logs/access-log
   </VirtualHost>

   <VirtualHost trainingpages.gbdirect.co.uk>
   ServerAdmin webmaster@gbdirect.co.uk
   DocumentRoot /home/www/web.trainingpages/docs
   ServerName trainingpages.gbdirect.co.uk
   ErrorLog /home/www/web.trainingpages/logs/error-log
   TransferLog /home/www/web.trainingpages/logs/access-log
   </VirtualHost>
   ```

   Note that the two names given `llord.gbdirect.co.uk` and `trainingpages.gbdirect.co.uk` should both resolve to 192.168.0.2

3. *Access Control*

   (a) You should create a file called `.htaccess` in both directories, the first should be:

   ```
   AuthType        Basic
   AuthName        "First Directory"
   AuthUserFile    /www/conf/auth/auth.user
   AuthGroupFile   /www/conf/auth/auth.group
   require user   testuser
   ```

   and the second should be:

   ```
   AuthType        Basic
   AuthName        "Second Directory"
   AuthUserFile    /www/conf/auth/auth.user
   AuthGroupFile   /www/conf/auth/auth.group
   require group   testgroup
   ```

Module 2

Key Configuration Files

Objectives

After completing this module, you should be able to configure the following:

- The password files `/etc/passwd`, `/etc/shadow`
- The group file `/etc/group`
- `cron` management `/etc/crontab`
- Kernel modules (`/etc/conf.modules`)
- Filesystem mounting (`/etc/fstab` and `/etc/exports`)
- System startup and shutdown scripts

2.1 `/etc/passwd`

- Stores information about users
  - ◇ Password (on some systems)
  - ◇ Id, and primary group
  - ◇ *Finger* information
  - ◇ Home directory
  - ◇ Default shell

2.2 `/etc/passwd` (continued)

- Colon-separated fields, e.g.

```
lee:Df18jed/nienysd:501:501:Lee Willis,Rm 1,013 567,013 765:/home/lee:/bin/bash
```

- First field is the username
- Second is the encrypted password [1]
- Third and fourth fields give the user ID and the primary group ID respectively
- *Finger* information is a comma separated list of information about a user
  - ◇ Typically stores full name, office room, office phone number and home phone number
- The sixth field is the user's home directory
- The user's default shell is given by the last field

[1]On systems which support shadow passwords this will just be an `x`, see 2.8 for an explanation

2.3 Editing `/etc/passwd`

- You should never edit `/etc/passwd` directly
  - ◇ Can lose information on multi-user systems
- Use the `passwd` command
- Normal users simply type `passwd`
  - ◇ Prompted for old password
  - ◇ Type new password twice (to avoid typos)
- Superuser can change anyone's password
  `passwd username`
  - ◇ Enters only the new password
  - ◇ Don't have to know the old password
- Superuser may also disable/enable accounts
  - ◇ `passwd -l username` disables or *locks* an account
  - ◇ `passwd -u username` *unlocks* the account

2.4 Other Changes To `/etc/passwd`

- `chfn` allows you to change the finger information for a user e.g.

```
$ chfn -f "Lee Willis" -o "Room 1" -p "01234 5678" -h "0123 45678"
```

- `chsh -s shell` lets you change your default shell
  - ◇ Must be listed in `/etc/shells`
  - ◇ `chsh --list-shells` will give a list of valid values
- Example:

```
$ chsh --list-shells
/bin/bash
/bin/sh
/bin/ash
/bin/bsh
/bin/tcsh
/bin/csh
$ chsh -s /bin/tcsh lee
Changing shell for lee.
Password:
Shell changed.
```

- *Note:* Both `chfn` and `chsh` require you to give your password

2.5 `/etc/group`

- Effective control of file access is one of the strengths of Linux/Unix
- One aspect of this is the concept of *groups*
- Users belong to one or more of these groups
- Access to files can be granted or denied on the basis of group privileges
- Group membership is controlled by the file `/etc/group`

2.6 Editing `/etc/group`

- Like `/etc/passwd` shouldn't be edited directly
- Tools can change it and ensure locking
- To *create* a group with ID *gid* and name *gname*:

  ```
  $ groupadd -g gid gname
  ```

- To change name of group *gname* to *newname*:

  ```
  $ groupmod -n newname gname
  ```

- `usermod` changes the groups a user belongs to,

  e.g. to add the user `lee` to groups `www`, `project`, and `tempgroup`:

  ```
  $ usermod -G www,project,tempgroup lee
  ```

- N.B. It also removes him from any groups not listed (excluding his primary group)
- `usermod` can also change the information in `/etc/passwd`
  - ◇ Can only be run by the superuser

2.7 Important Note

- Changing user information shouldn't be undertaken lightly
- There are a number of restraints on changing usernames, IDs, and group IDs
- You shouldn't change name while a user is logged in
- You shouldn't change ID while user has processes running
- See `man usermod` and `man groupmod` for others
- Mostly common sense

2.8 Shadow Passwords

- *shadow passwords* are a security feature
  - ♢ Normal users could get others' passwords if encrypted versions were readable
  - ♢ Some information in `/etc/passwd` needs to be readable, but *Passwords* don't!
- Solution:
  - ♢ Keep everything except passwords in `/etc/passwd`
  - ♢ Password field contains just a single *'x'*
- Encrypted passwords are stored in `/etc/shadow`
  - ♢ Only readable by superuser

2.9 `/etc/shadow`

- `/etc/shadow` also stores other information
- Mainly password expiry information
- Can force users to change their password
- Most important benefit is increased security
- All modern systems should use shadow passwords

2.10 Scheduling Jobs (`Cron`)

- `cron` schedules jobs to run at times; specified in the file `/etc/crontab`

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily

5,35 * * * 1-5 root /usr/local/bin/domail
```

- The first section sets environment variables
- Cron jobs run when the current time/date matches a `crontab` entry
- The first 5 fields in `/etc/crontab` are

```
minute hour day_of_month month day_of_week
```

- `*` Matches all possible values
- Commas separate sets of values within a field
- Ranges can also be specified, e.g. `1-5`
- Can also specifiy steps, e.g. `0-59/5`

2.11 `/etc/crontab`

- `/etc/crontab` also specifies what *user* the job runs as, e.g.

```
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
```

- Second line runs the command `run-parts /etc/cron.daily` as `root` at 4:02am every day
- To run the command `/usr/local/bin/domail` as `root` *at* 10 minutes past *and* 40 minutes past the hour, *between* 9am and 6pm *on weekdays*:

```
10,40 9-18 * * 0-5 mail /usr/local/bin/domail
```

2.12 `run-parts`

- `run-parts` is a script designed for use with `cron`
- Runs all the programs in the specified directory
- Allows administrators to easily add jobs
  - ♢ Simply place an executable script/program in the correct directory
- N.B. *Not* a standard cron feature

2.13 `logrotate`

- Log rotation is normally handled by `logrotate`
- Run by cron, which reads `/etc/logrotate.conf` for configuration
- Example:

```
$ cat /etc/logrotate.conf
#Rotate the logs weekly
weekly
# keep 4 weeks worth of backlogs
rotate 4
# send errors to root
errors root
# create new (empty) log files after rotating old ones
create
# uncomment this if you want your log files compressed
#compress
# RPM packages drop log rotation information into this directory
include /etc/logrotate.d
# no packages own lastlog or wtmp -- we'll rotate them here
/var/log/wtmp {
    monthly
    rotate 1
}
/var/log/lastlog {
    monthly
    rotate 1
}
```

2.14 Module Configuration

- The Linux kernel can be modular in nature
- Needs to know which devices use which drivers
- `/etc/conf.modules` contains this information [2]
- Typical file may look like :

```
alias eth0 ne2k-pci
alias eth1 3c509
```

- States that the device `eth0` requires the module `ne2k-pci`, and `eth1` requires `3c509`

[2] Warning, on some systems this is `/etc/modules.conf`

2.15 Modules Configuration - ‘Options’

- Some modules allow you to specify options
- Mainly used for ISA peripherals, e.g. to provide I/O and IRQ information:

  ```
  alias eth0 ne
  options ne irq=10
  ```

  Specifies that `eth0` requires the module `ne` which should be passed the argument `irq=10`
- Can also specify actions to be executed when loading unloading modules, e.g.

  ```
  pre-install pcmcia_core /etc/rc.d/init.d/pcmcia start
  ```

  Run `/etc/rc.d/init.d/pcmcia start` before loading the `pcmcia_core` module

2.16 Mounting Filesystems

- Linux can store its files on multiple disks
- It decides what part of the filesystem each of these lives on using `/etc/fstab`

Logical Volume	Mount Point	FS type	Options	Dump	Check order
/dev/hda1	/	ext2	defaults	1	1
/dev/hda5	/home	ext2	defaults	1	2
/dev/hda7	/tmp	ext2	defaults	1	2
/dev/hda6	/usr	ext2	defaults	1	2
/dev/hda8	swap	swap	defaults	0	0
/dev/fd0	/mnt/floppy	ext2	noauto	0	0
/dev/cdrom	/mnt/cdrom	iso9660	noauto,ro	0	0
\\kashmir\c	/mnt/kashmir	smbfs	guest	0	0
landlord:/var/admin	/var/admin	nfs	defaults	0	0
landlord:/home/lee	/home/lee/LANDLORD	nfs	defaults	0	0

2.17 Runlevels

- Linux has several modes of operation
- Referred to as *runlevels*
- Most common are:

0	Initial boot
1	Single User Mode
2 and above	Multi-user mode

- Apply to most UNIX/Linux, but some allocate different numbers to graphical mode/login [3]
- Unfortunately, the app and daemons run automatically at each level vary greatly
- A good justification for the Linux Standard Base
- See your distribution documentation for details

[3]Red Hat uses 5 for graphic mode, but others don't.

2.18 Single User Mode

- Mainly used for diagnostic purposes
- Starts only a subset of the possible services, e.g.
  - ♢ No networking
  - ♢ No mail services
  - ♢ No name lookup services
    - ∗ Except `/etc/hosts`
  - ♢ No file-sharing services etc

2.19 Multi User Mode

- The ‘normal’ operating state
- All configured services are running
- Multiple users can log in
- `/sbin/runlevel` shows the previous and current runlevel of your machine

2.20 Starting up and Shutting down

- Only the *superuser* can shutdown or reboot
- `halt` will shut down the machine totally
  - ♢ For safety you should type `/sbin/halt`
- Makes sure all processes are stopped
- Stops services cleanly
- Writes unsaved data to the disk
  - ♢ ‘Syncing’
- `/sbin/reboot` will shut down cleanly and reboot

2.21 Changing runlevel

- It is sometimes necessary to change runlevel
- Rare, but useful to know
- You can instruct a system to change runlevel using the `telinit` command
- Example:

```
$ telinit 5
```

- Changes to runlevel 5
- `telinit 1` takes the system down to single user mode

2.22 Initscripts

- The precise behaviour of each of the runlevels is controlled by *initscripts*
- Control which services run in each runlevel
- Live in `init.d`
  - ♢ On Debian it's in `/etc/init.d`
  - ♢ On Redhat it's `/etc/rc.d/init.d`
- Each file here is a script that can be called with an argument, `start`, `stop`, or `restart`

2.23 `rcn.d`

- The contents of the directories `rc`*`n`*`.d` control which services start and stop in runlevel *n*
- The directories hold symbolic links to the files in `init.d`
- The links are named informatively
- To start service *abc* you would create a link typically named `Sxx`*`abc`*, to `init.d/`*`abc`*
- The `xx` specifies the order to run the scripts, e.g. `S00foo` will be run before `S90foo`
- Links that stop a service are of the form `Kxx`*`abc`*

2.24 Initscripts - An example

- Consider the following: [4]

```
lee @ 12:22:08 /etc/rc.d/rc3.d ls -l S*

lrwxrwxrwx ... S01kerneld -> ../init.d/kerneld
lrwxrwxrwx ... S10network -> ../init.d/network
lrwxrwxrwx ... S15nfsfs -> ../init.d/nfsfs
lrwxrwxrwx ... S20random -> ../init.d/random
lrwxrwxrwx ... S30syslog -> ../init.d/syslog
lrwxrwxrwx ... S40atd -> ../init.d/atd
lrwxrwxrwx ... S40crond -> ../init.d/crond
lrwxrwxrwx ... S40portmap -> ../init.d/portmap
lrwxrwxrwx ... S40snmpd -> ../init.d/snmpd
lrwxrwxrwx ... S45pcmcia -> ../init.d/pcmcia
lrwxrwxrwx ... S50inet -> ../init.d/inee
lrwxrwxrwx ... S55named -> ../init.d/named
```

- We can see that the first thing started is `kerneld`, followed by `network` services, `nfs` services, etc
- There are also a series of `Kxxyyy` scripts which shut down the services in a sensible order

[4]Unimportant information has been removed from the screen dump so do not be alarmed if this doesn't look like you'd expect!

2.25 Restarting Services

- Can be necessary to restart a particular service, e.g. so it can re-read a modified configuration file
- This can be done without a complete reboot
- It must, however, be done by the superuser
- To restart samba (`smb`) we can do the following:

```
$ cd /etc/rc.d/init.d
$ ./smb restart
```

2.26 Exercises

1. *Passwords*
   (a) Find out whether your machine is using standard or shadow passwords?
2. *Users*
   (a) Add a new user (`useradd`) and set them up with the correct Full Name, password, home directory. Set their default shell to `csh`
3. *Groups*
   (a) Create a new group and add your user to this group
   (b) Now remove both the user and the group. How would you ensure that all files belonging to that user have been removed?
4. *Scheduling*
   (a) Add a cron job to eject your CDROM drive at 5 minutes past every hour and put it back in at ten minutes past the hour
5. *Mounting*
   (a) Set up your `fstab` so that

   ```
   $ mount /dev/cdrom
   ```

   will automatically mount your CD drive under `/mnt/cdrom`
6. *Runlevels*
   (a) Switch your machine between runlevels 3 and 5. What is happening? What happens if you change to runlevel 6?
   (b) Make sure your machine runs the same set of services in both runlevels
7. *Stop, Start and Restart Services*
   (a) Check you can stop, start, or restart services
   (b) Can you do this as a normal (ie non-root) user?

2.27 Solutions

1. *Passwords*

   (a) Your machine will have an `/etc/shadow` file if it is using shadow passwords. The password field will be set to an 'x' in `/etc/passwd`.

2. *Users*

   (a) The following would set the details for the user Lee Willis

   ```
   $ useradd leewillis
   $ passwd leewillis
   Changing password for user leewillis
   New UNIX password:
   Retype new UNIX password:
   passwd: all authentication tokens updated successfully
   $ chfn -f "Lee Willis" leewillis
   Changing finger information for leewillis.
   Finger information changed.
   $ chsh -s /bin/csh leewillis
   Changing shell for leewillis.
   Shell changed.
   ```

   The home directory should be set up properly (`/home/leewillis`), if not you can change it with

   ```
   $ usermod -d /home/leewillis leewillis
   ```

3. *Groups*

   (a)
   ```
   $ groupadd newgroup
   $ usermod -G newgroup leewillis
   ```

   (b) To remove the group, the user and the user's home directory

   ```
   $ groupdel newgroup
   $ userdel -r leewillis
   ```

   There are a few important points here! Firstly there may still be files in the filesystem belonging to that user. To locate them all you should have done

   ```
   find / -user leewillis -exec rm -f {} \;
   ```

   prior to removing the user. You should also have located all files belonging to the group and re-parented and/or removed them before removing the group

4. *Scheduling*

   (a) The following lines should acheive the desired effect

   ```
   05 * * * * root eject /dev/cdrom
   10 * * * * root eject -t /dev/cdrom
   ```

5. *Mounting*

   (a) The entry should look like

   ```
   /dev/cdrom       /mnt/cdrom       iso9660 noauto,ro     0 0
   ```

6. *Runlevels*

   (a) You can change runlevels by using `telinit 5` and `telinit 3`. All non-relevant services are stopped and the new ones started each time you change runlevel. Runlevel 6 reboots the machine!
   (b) You should ensure that the directory listings for `/etc/rc.d/rc3.5` and `/etc/rc.d/rc5.d` are the same. This should ensure that the same services are started/stopped when entering either runlevel.

7. *Start, Stop and Restart services*
   (a) -
   (b) -

Module 3

Dial Up and Remote Access

Objectives

On completion of this module, you should be able to:

- Understand the principles of point-to-point networking
- Understand and use the key protocols involved in dial-up networking
- Configure Linux as a dial-up server
- Configure Linux as a dial-up client

3.1 Dial-In/Out

- Linux can be used both as a dial-up client and a server
- A client system can be used to dial out to another system
- A server accepts dial-in
- There are a myriad of options for doing this
- We look at a very limited set to get you going
- The PPP Howto covers much more detail (try `/usr/doc/HOWTO/PPP-HOWTO`)

3.2 The Basics

- The standard for point-to-point links such as dial-up is PPP
- This is the point-to-point-protocol; implemented in Linux through `pppd`
- Very widely used indeed
- Supersedes the older and now retired SLIP (serial line IP) protocol
- Can dynamically negotiate local and remote addresses plus much more
- Standard for IP over leased lines and connections to ISPs etc.
- All normal distributions come with support
- Requires kernel support; modern distributions all provide the relevant support in the kernel they ship

3.3 Authentication

- PPP can use CHAP or PAP or nothing at all
- PAP is the standard login/password mechanism (Password Authentication Protocol)
- PAP is not the most secure
- CHAP involves regular ‘challenges’ and ‘responses’
  - ♢ Each side knows ‘secrets’ that can be used to encrypt challenges and responses
  - ♢ We leave reading up as an exercise
  - ♢ The nothing at all option relies on just a login/password being used to authenticate the dialup login

3.4 Setting-up dial-out

- Vast and complicated range of options
- In essence, must arrange for a call to be placed and `pppd` started
- Normally uses the `chat` command to place the call (talk to modem etc)
- Then starts `pppd` to handle the link
- `kppp` or `wvdial` provide easier configuration
- Particular difficulty if you have multiple ISPs
  - ♢ `/etc/resolv.conf` indicates your nameservers
  - ♢ but these differ for each ISP
  - ♢ `kppp` handles this (amazing)

3.5 One Dial-Out Setup

- In ˜/user/.ppprc

```
detach
modem
crtscts
lock
:192.168.0.142
proxyarp
```

- Use system config tools, e.g. RedHat's control panel or SuSE's `yast` to add ppp0 interface

3.6 Dial-Out Setup contd.

- in `/etc/sysconfig/network-scripts/chat-ppp0`

```
'ABORT' 'BUSY'
'ABORT' 'ERROR'
'ABORT' 'NO CARRIER'
'ABORT' 'NO DIALTONE'
'ABORT' 'Invalid Login'
'ABORT' 'Login incorrect'
'TIMEOUT' 90
'' '\d+++\dATH0Z&d0&c1M0'
'OK' 'ATDTnnnnnnnnnnn'
'CONNECT' ''
'ogin:--ogin:' 'pppmikeb'
'ssword' 'mikebpasswd'
'}' ''
```

- To dial: `/sbin/ifup ppp0` (as root)

3.7 Dial-In Setup

- in `/etc/conf.getty.ttyS2`

```
DEBUG=010
INIT="" \d+++\dATH0\r OK\r\n AT\sM1\sE1\sQ0\sV1\sX4\sS0=1\r OK\r\n
DELAY=1
TIMEOUT=30
```

- in `/etc/inittab`

```
m2:2345:respawn:/sbin/getty ttyS2 F57600
```

- in `/etc/passwd`

```
pppmikeb:xxx:501:100:Mike Banahan,,,,:/home/mikeb:/usr/sbin/pppd
```

3.8 Other options

- worth checking the documentation on `diald` if you want dial-on-demand
- if possible, use `kppp` to set up and manage your connections; similarity to win95 is strong and the model is good
- bear in mind that networking is a system, not a user feature: the Windows model that users can start networking is not appropriate for servers
- This is generally a tricky area to set up. Allocate a full day.
- May be able to do some debugging with `cu`

  ```
  cu -l /dev/ttyS2 dir
  ```

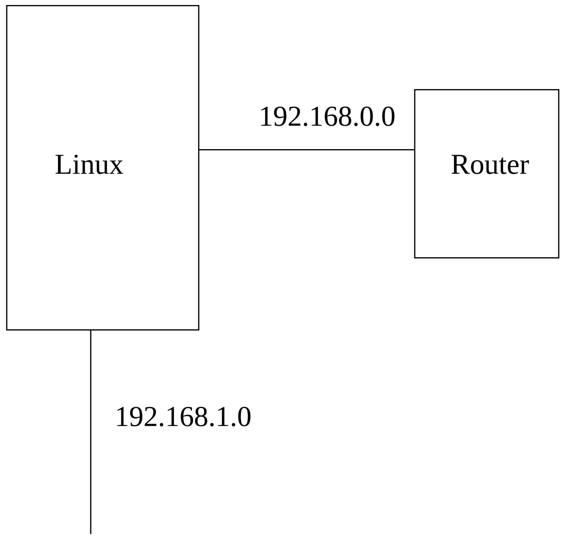
- Sorry if it's not easy
- Even better option
  - ♢ For remote dial-in, consider using ISDN and routers
  - ♢ makes it laughably simple

3.9 Preferred Installation Architectures

- There are any number of ways of setting up mail and web access
- Will usually use Linux system as an intermediary
  - ◇ `sendmail` for mail exchange
  - ◇ `squid` as proxy
  - ◇ possibly using firewalling to limit access to ports
  - ◇ possibly using IP masquerading for specific services if no proxy available

3.10 Preferred Setup 1

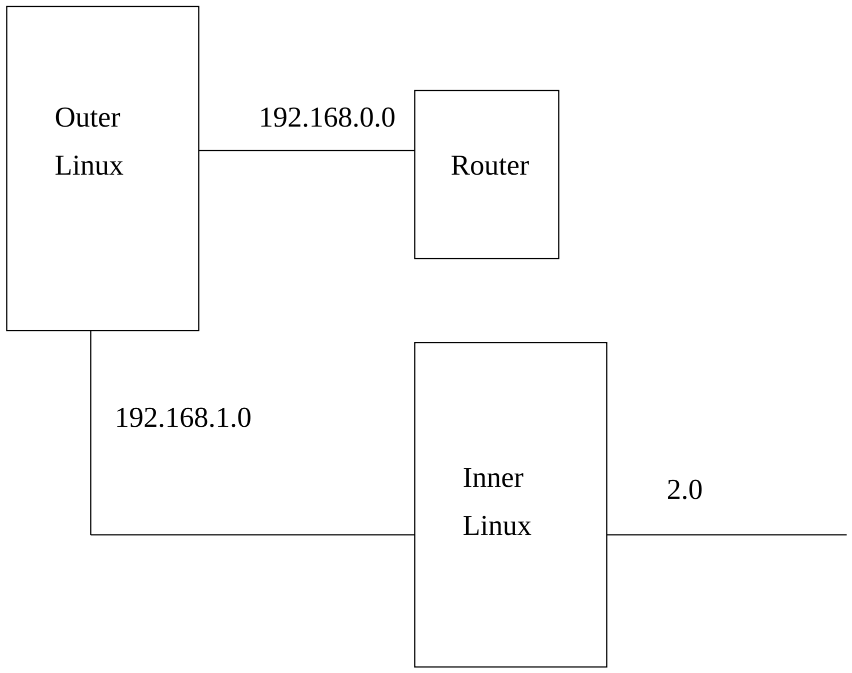
- As described above
- Internal network uses private addresses such as 192.168.10.0
- External network uses further interface and private address
- Dial-on-demand router used for ISDN access to ISP
- DNS is run on Linux system with zone files for internal domains
- `sendmail` set up *not* to query DNS for local email [1]
- `fetchmail` or `ping` used to connect to ISP occasionally for inbound mail



[1]The basics of this go deep into sendmail configuration. In essence, the `Oc` option must be set for sendmail, the relay should be marked as 'expensive' and one rule set must be commented out to prevent routine DNS lookups from causing outbound dial-up. The details are appended to this section.

3.11 Preferred Setup 2

- As described above
- If internal network addresses already ’illegal’ (i.e. allocated in real world)
- Double-proxy essential services with Squid, Sendmail etc
- Alternative is to use IP masquerading (but that has problems) [2]
- Inner Squid/Sendmail simply forward to outer
- Outer Sendmail delivers to inner
- In all configurations, router can equally be to leased line
  - ♢ in which case can run Apache too



[2]Masquerading looks attractive but it fails with protocols that embed IP addresses (as some do). It is not a panacea and proxies may *have* to be used.

3.12 Appendix - sendmail configuration for non dial-on-demand

- You must mark the relay mailer as expensive with the ’e’ flag:

```
Mrelay,   P=[IPC], F=emDFMuXa8, S=11/31, R=61, E=\r\n, L=2040
```

- In the options section of `sendmail.cf` you must insert

```
Oc
```

- To stop the machine attempting name lookups for local mail you should comment out a section in /etc/sendmail.cf

```
# pass to name server to make hostname canonical
#R$* < @ $* $~P > $*           $: $1 < @ $[ $2 $3 $] > $4
```

Module 4

Email

Objectives

By the end of this section you should :

- Understand how email works
- Be familiar with the basics of sendmail
- Be able to add new accounts and aliases

4.1 How email works

- Messages in transit handled by Mail Transport Agents (MTAs)
  - ◊ Sendmail
  - ◊ Exim
  - ◊ Qmail
- Resposible for passing a message from one machine to another
- Mail sent and read using Mail User Agents (MUAs)
  - ◊ Outlook express
  - ◊ Netscape
  - ◊ Pine

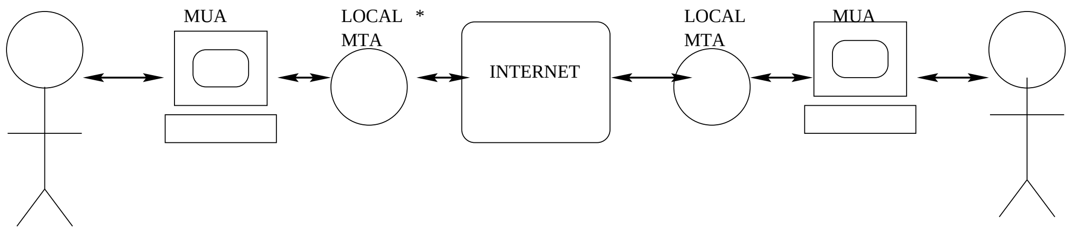
4.2 Where does an email go?

- Routing of messages is dependent on your domain
- Mail services for a given domain are advertised via the DNS service
  - ♢ Mail Exchanger (MX) records
  - ♢ May be more than one for a given domain

```
$ dig mx bbc.co.uk
....
;; ANSWERS:
bbc.co.uk.      1800    MX      10 relay1.pipex.net.
bbc.co.uk.      1800    MX      20 relay2.pipex.net.
bbc.co.uk.      1800    MX      5 mail.bbc.co.uk.
bbc.co.uk.      1800    MX      7 mailb.bbc.co.uk.
....
```

- Tried in priority order (lower is better)
  - ♢ 5->7->10->20
- What happens after MX gets your message is setup dependent
- Simplest case, message sits on your server until you check
- May travel from company server to departmental server etc.

4.3 Overview

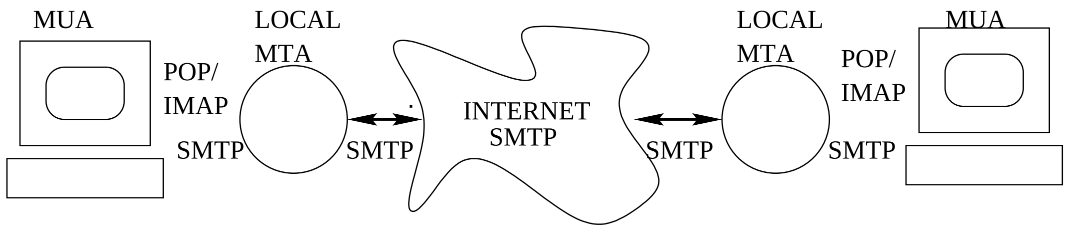


* NOT ALWAYS "LOCAL". OFTEN AT ISP SUCH AS DEMON OR FREESERVE.

4.4 Email Protocols

- Number of protocols for transfer of email
  - ♢ SMTP (Simple Mail Transfer Protocol)
  - ♢ POP (Post Office Protocol)
  - ♢ IMAP (Internet Message Access Protocol)
- SMTP used to communicate between MTAs
  - ♢ Also ESMTP - Extended SMTP
  - ♢ Delivery Status Notification
  - ♢ 8-Bit MIME messages
- POP/IMAP communicate between mail servers and MUAs

4.5 Where the protocols go



4.6 Basic Installation

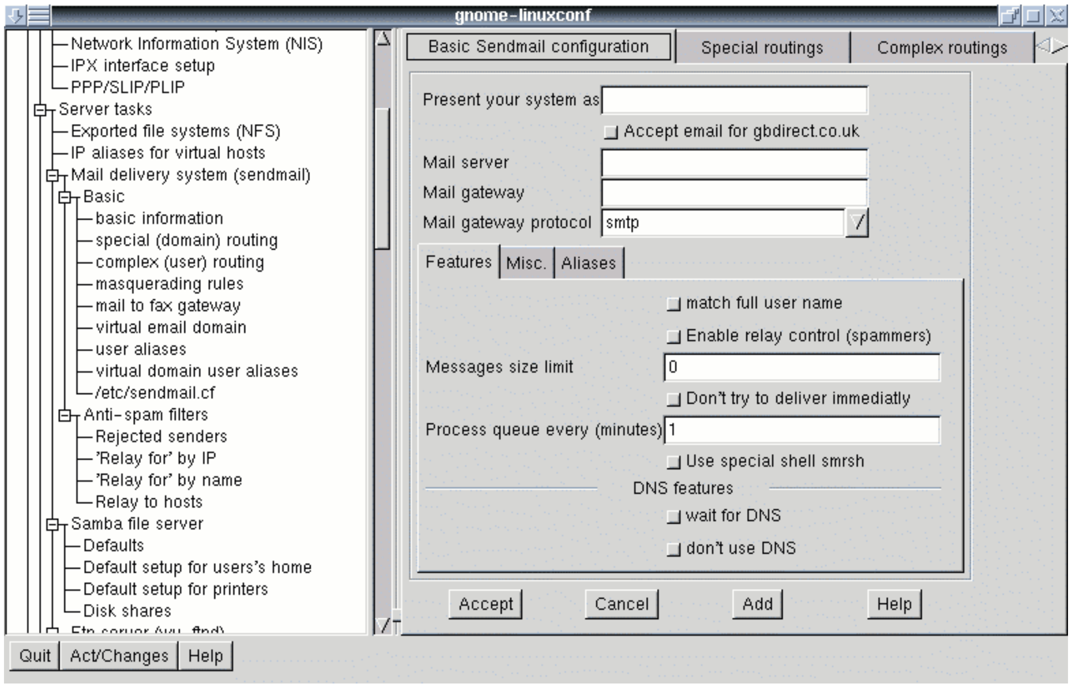
- Most common implementation is to run sendmail
- Installed by Redhat by default
  - ◊ Exim becoming more popular
  - ◊ Sendmail battle tested though configuration can be nasty
  - ◊ Exim easier to configure
  - ◊ Not yet battle-tested ...
- GUI config tools know about sendmail
  - ◊ Linuxconf
- Main configuration file `sendmail.cf`
  - ◊ Not pleasant
  - ◊ B. Costales (1997) Sendmail, London: O'Reilly. for around 750 more pages of detail

4.7 Sendmail behaviour

- Commonly run in one of two setups
  - ◇ Immediate send
  - ◇ Queue then send
- Immediate send is useful with permanent net connections
  - ◇ Send and receives email
  - ◇ Runs permanently
- Queueing useful for dial-up connections
  - ◇ Sendmail queues outgoing mail
  - ◇ Sends every x minutes
  - ◇ `sendmail -q60m` [1]
  - ◇ Still have sendmail running permanently, to accept email and queue it

[1] Redhat set this up in `/etc/sysconfig/sendmail`

4.8 Linuxconf & sendmail



Present your system as

☐ Accept email for gbdirect.co.uk

Mail server

Mail gateway

Mail gateway protocol smtp

☐ match full user name

☐ Enable relay control (spammers)

Messages size limit 0

☐ Don't try to deliver immediatly

Process queue every (minutes) 1

☐ Use special shell smrsh

DNS features

☐ wait for DNS

☐ don't use DNS

4.9 Configuration files (`sendmail.cf`)

- Most important bits are probably :

```
#file containing name of hosts we accept mail for
Fw/etc/sendmail.cw
# my official domain name
Djgbdirect.co.uk
# "Smart" relay host (may be null)
DSmailgate.gbdirect.co.uk
# who I masquerade as (null for no masquerading) (see also $=M)
DMgbdirect.co.uk
# file containing IP numbers of machines which can use our relay
F{LocalIP} /etc/mail/ip_allow
# file containing names of machines which can use our relay
F{LocalNames} /etc/mail/name_allow
# file containing names we relay to
F{RelayTo} /etc/mail/relay_allow
```

- Also contains numerous options such as
  - ◇ `Oc` - Don’t connect to expensive mailers
  - ◇ `O LogLevel` - Determine the level of logging
- Sendmail must be restarted after changing `sendmail.cf` [2]

[2] On Redhat, running `/etc/rc.d/init.d/sendmail restart`

4.10 Configuration files (`sendmail.cw` and `/etc/mail/`)

- `sendmail.cw`

```
# sendmail.cw - include all aliases for your machine here.
acronym.co.uk
gbdirect.co.uk
trainingpages.co.uk
uklc.org
```

- `/etc/mail/ip_allow`

```
192.168.0
```

4.11 Explanation of examples

- Anyone can talk to this server to deliver mail to users at:
  `acronym.co.uk`
  `gbdirect.co.uk`
  `trainingpages.co.uk`
  `uklc.org`
- Machines on the `192.168.0` network can use the server to send outbound email
- Outbound messages will appear to come from `user@gbdirect.co.uk` instead of `user@beehive.gbdirect.co.uk` etc.
- Messages we can’t deal with (ie non-local emails) will be forwarded to `mailgate.gbdirect.co.uk`

4.12 Monitoring sendmail

- Most distributions log sendmail messages to `/var/log/maillog`
- Details of emails sent and received

```
Oct  2 15:01:20 landlord sendmail[18735]: PAA18735:
from=<owner-linux-kernel-outgoing@vger.rutgers.edu>, size=2636,
class=-60, pri=140636, nrcpts=1, msgid=<Pine.LNX.4.05.991007135
6340.32297-100000@ns.snowman.net>,
proto=ESMTP, relay=root@samosa.gbdirect.co.uk [192.168.0.143]
Oct  2 15:01:21 landlord sendmail[18736]: PAA18735:
to=<lee@gbdirect.co.uk>, delay=00:00:01, xdelay=00:00:01,
mailer=local, stat=Sent
```

- Logs the date, machine name, process ID, Queue ID
- Mails in the queue commonly stored under `/var/spool/mqueue`
- Location can be changed in `/etc/sendmail.cf`
- Multiple lines per message, source and one or more destinations.
  - ♢ Source shows size of message, Message-ID, protocol used
  - ♢ Destinations shows delay in delivering if any, delivery method (mailer) and status (stat)
  - ♢ Also commonly logs POP/IMAP connections from local clients

```
Oct  2 15:01:06 landlord ipop3d[18733]:
Login user=julie host=kashmir.gbdirect.co.uk
Oct  2 15:01:08 landlord ipop3d[18733]:
Logout user=julie host=kashmir.gbdirect.co.uk
```

4.13 Monitoring sendmail (cont.)

- `mailq` can be used to see what messages are awaiting delivery
  - ◇ Alias for `sendmail -bp`
- Shows lots of information
  - ◇ Time Queued
  - ◇ Size of Message
  - ◇ Source Address
  - ◇ Destination Address
  - ◇ Status
  - ◇ ID in Queue

4.14 Example of `mailq`

```
$ /usr/lib/sendmail -bp
                Mail Queue (5 requests)
--Q-ID-- --Size-- -----Q-Time----- --Sender/Recipient------
OAA15282   551674 Thu Oct  7 14:33 www
                                   novak@arcon.cz
AAA07174      113 Thu Oct  7 00:34 <janes27812@ragemail.com>
                 (host map: lookup (ragemail.com): deferred)
                                   <mikeb@acronym.co.uk>
NAA14290* 1277047 Thu Oct  7 13:36 www
                                   sealserv@nbnet.co.ke
NAA14327  3005358 Thu Oct  7 13:37 www
                                   td@inkjet.se
NAA14293  3005358 Thu Oct  7 13:36 www
                                   sealserv@nbnet.co.ke
```

4.15 Talking the talk ...

- SMTP, POP, and IMAP all human-readable protocols
- Can talk directly to servers
- Telnet to applicable port
  - ♢ SMTP - 25
  - ♢ POP - 110
  - ♢ IMAP - 143

4.16 Talking SMTP

```
$ telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 beehive.gbdirect.co.uk ESMTP Sendmail 8.9.3/8.9.3;
Fri, 1 Oct 1999 09:23:11 +0100
MAIL FROM: lee@gbdirect.co.uk
250 lee@gbdirect.co.uk... Sender ok
RCPT TO: lee
250 lee... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
This is a test message
Body of email goes here
.
250 JAA00609 Message accepted for delivery
quit
221 beehive.gbdirect.co.uk closing connection
Connection closed by foreign host.
```

4.17 Talking POP

```
$ telnet localhost 110
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
+OK POP3 localhost v4.47 server ready
USER lee
+OK User name accepted, password please
PASS ******
+OK Mailbox open, 1 messages
LIST
+OK Mailbox scan listing follows
1 329
.
RETR 1
+OK 329 octets
Return-Path: <lee>
Received: (from lee@localhost)
        by gbdirect.co.uk (8.8.7/8.8.7) id JAA25308
        for lee; Fri, 8 Oct 1999 09:39:07 +0100
Date: Fri, 8 Oct 1999 09:39:07 +0100
From: Lee <lee@gbdirect.co.uk>
Message-Id: <199910080839.JAA25308@gbdirect.co.uk>
To: lee@gbdirect.co.uk
Subject: foo
Status:
bar
baz
bat
.
DELE 1
+OK Message deleted
QUIT
+OK Sayonara
Connection closed by foreign host.
```

4.18 Managing Users under sendmail

- Users on a box "just work"

  ```
  $ useradd newuser
  $ passwd newuser
  New UNIX password:
  Retype new UNIX password:
  passwd: all authentication tokens updated successfully
  $ mail newuser < /tmp/welcome_msg
  ```

- Sometimes desirable to have "aliases" for users
- Mail for a certain address actually gets delivered to a different address
  - ◊ Or to more than one address!

```
JohnS@dom.co.uk -> John22@dom.co.uk
management@dom.co.uk -> lisa@dom.co.uk & peter@otherdom.co.uk
sales@dom.co.uk -> amanda@dom.co.uk & john@dom.co.uk
```

4.19 `/etc/aliases`

- Controls email aliases
  - ♢ Human readable
  - ♢ *MUST* run 'newaliases' before sendmail will notice changes
- E.g.

```
JohnS:          John22
management:     lisa, peter@otherdom.co.uk
sales:          amanda, john
```

4.20 Aliases - other uses

- Can also use the alias mechanism for logging information to a file
- Useful for company wide enquiries, e.g.

```
/etc/aliases:
contact_file:   /usr/maillogs/contact
contact_users:  mikeb, julie, lee, davef
contact:        contact_file, contact_users
```

- Logs the message to the end of the file `/usr/maillogs/contact` and sends it to `mikeb`, `julie`, `lee`, `davef`

4.21 Sendmail Exercises

1. *Setup your system to use sendmail as daemon*
   (a) Check that it starts correctly
   (b) Telnet to port 25 and check that it delivers mail to a local recipient.
2. *Telnet to port 25 and try to use sendmail as a relay*
   (a) Ensure that it correctly bars relaying
   (b) Switch on relaying for a particular domain and ensure that it works
3. *Create suitable entries in* `/etc/aliases`
   (a) So that mail addressed to one name is sent to another
   (b) So that mail addressed to that name is archived to a file
   (c) So that mail addressed to that name is piped into a shell script of your choice

Module 5

Basic Filesystem

Objectives

- After completing this section, you will be able to:
  - ♢ Understand a typical Linux filesystem
  - ♢ Navigate the file hierarchy
  - ♢ Manipulate files and directories
  - ♢ Handle access control
  - ♢ Deal with ‘special files’ and *links*

5.1 Filesystem Overview

- Linux uses *ext2* as its *native* filesystem
  - ♢ Also supports many other types
- All data stored on a Linux system is a file
- Ext2 file names can be 1 to 255 characters long
  - ♢ Only */* and *nul* are disallowed
- Non-native filesystems have different features
- Ext2 sees only two basic types of files:
  - ♢ *directories*
  - ♢ *files*
- Other specialised types exist (FIFOs, and ‘special files’), these are covered later

5.2 Files

- Linux imposes no structure on files
- All files are accessible at the byte level
- Individual files have a maximum size of around 2Gb (*in an ext2 filesystem*)
- They have a minimum size of 0 bytes
- Files can be extended after creation
- Filename extensions such as *.exe* and *.bat* are unnecessary
- Executable files are simply marked as such using file permissions (*see later*)

5.3 Directories

- *Directories* are files that list other files
  - ♢ Can be normal files or directories
  - ♢ Enables a hierarchy to be built
- Each directory entry consists of two parts: a file name and an *inode number*

  (An inode is roughly a *pointer to a file*, see below)

Filename	Inode number
.	512
..	500
bin	17324
basic_linux.tex	24567

- The topmost directory is always called `/`
  - ♢ Called the fileystem ‘root’
- Directory information can only be changed by Linux itself
  - ♢ Ensures a proper structure is maintained

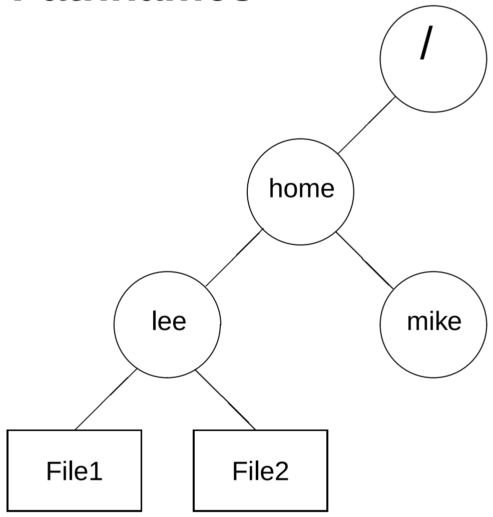
5.4 Directory Hierarchy

- By tradition several directories have specialised rôles

`/bin`	executable commands
`/sbin`	executable commands regarding important system functions
`/etc`	system configuration files
`/lib`	shared libraries
`/dev`	peripheral devices
`/tmp`	temporary files
`/mnt`	to mount external devices
`/var`	odds and ends, e.g. logs, status and lock files, spooling files
`/proc`	system information
`/usr/bin`	further executable files
`/usr/sbin`	further system-important executable files
`/usr/lib`	further libraries

- User-installed programs typically go under the `/usr/local` hierachy
- `/mnt` is not always present, it is merely a convenience to place all ‘mounted’ devices under one place

5.5 Pathnames



- Files can be referred to by *relative* or *absolute* pathnames
- Absolute pathnames begin with `/`

```
/usr/sbin/httpd
/usr/local/bin/safe-mysqld
```

- The *absolute* pathname refers to one file only
- A relative pathname does not begin with `/` and describes the path from the current directory to find a file, e.g.

```
sbin/httpd
bin/safe-mysqld
```

5.6 Current Directory

- When you log in your shell is placed in your *home directory*
  - ◇ Typically `/home/username`[1]
  - ◇ Superuser typically has `/root` for a home directory
- `~` is a synonym for `/home/username`
- `cd` changes your current directory
  - ◇ Typing `cd path` changes your current directory to `path`
- `path` can be *absolute* or *relative*
- Without arguments `cd` changes to your home directory
- `pwd` tells you the current directory

[1]Home directories are sometimes under `/usr/username` on some older systems

5.7 Dot (.) and DotDot(..)

- Directories always contain two entries "." and ".."

`.`	Current directory
`..`	Parent directory

- Used for relative pathnames and navigation
- Example:

`$ mv file ..`	Move `file` to the parent directory
`$ du .`	Display space used by files in current directory and below
`$ du ..`	Display space used by files in parent directory and below
`$ ./a.out`	Execute the file `a.out` in the current directory

- This last row above shows forced execution of a particular file
  - ◊ If we had simply typed `a.out` then our *PATH* environment variable would be used to search for the file
  - ◊ It may execute another `a.out` instead of the one we *actually* want

5.8 Moving and Copying Files

- The `mv` command is used to move files:

  ```
  mv [options] source dest
  mv [options] source... directory
  ```

  e.g.

  ```
  $ mv oldname newname
  $ mv somefile ..
  ```

- The `cp` command is used to copy files:

  ```
  cp [options] source dest
  cp [options] source... directory
  ```

  e.g.

  ```
  $ cp thisfile newfile
  $ cp file1 file2 file3 /tmp
  ```

- Full details of applicable options and usage can be found by typing `man mv` or `man cp`

5.9 Removing Files

- Files are removed using the rm command:

  ```
  rm [options] file
  ```

  e.g.

  ```
  $ rm thisfile thatfile
  rm: remove ‘thisfile’? y
  rm: remove ‘thatfile’? y
  $
  ```

- Most notable among the options are :

`rm -f`	Force removal, without confirmation
`rm -r`	Recursively delete files

  e.g.

  ```
  $ rm -r somedir
  ```

- Again, full details are available by typing `man rm`
- Removing a file is *not* considered an operation on the file
  - ♢ It is an operation on the directory
  - ♢ Filenames are merely *links* (Explained below)

5.10 Operations on Directories

`mkdir`	Create a new directory
`rmdir`	Remove a directory
`ls`	List the contents of a directory

- These commands can take many arguments
- `mkdir` can be told to create the whole pathname of directories if they don't exist, e.g.

  ```
  $ mkdir -p /home/lee/new1/test/directory
  ```

  Will create the directories `/home/lee/new1` and `/home/lee/new1/test` as well as `/home/lee/new1/test/directory` if they don't already exist
- `ls` arguments control what information is shown and how it's sorted
  - ◇ Some are explained later, consult `man ls` for full details

5.11 Inodes

- Each file is represented by an inode[2]
- An *inode* contains information about:
  - ◇ File type (ordinary, directory, FIFO, block device etc.)
  - ◇ Owner ID (user the file belongs to)
  - ◇ Size (in bytes)
  - ◇ Access, creation, and modification times
  - ◇ Group ID (group the file belongs to)
  - ◇ File permissions
  - ◇ Mapping of the file contents (data sectors)
- Inode layout and location varies with filesystem type

[2]The term *inode* was invented by Dennis Ritchie of AT&T. He admits to forgetting why he chose that name.

5.12 Inodes (*continued*)

- `ls -i` displays inode numbers of entries, e.g.

```
200808 Ext2fs-0.1-14.html   542726 include
200795 Ext2fs-0.1-2.html    188447 info
200797 Ext2fs-0.1-4.html    333831 ldap
200802 Ext2fs-0.1-8.html    329729 man
200803 Ext2fs-0.1-9.html    278533 misc
200793 Ext2fs-0.1.html      428042 nsmail
204802 systemprogramming
```

- `stat` prints the inode contents for files inc. permissions, size, links, access times etc.

```
$ stat /home/lee
File: "/home/lee"
Size: 4096         Filetype: Directory
Mode: (0755/drwxr-xr-x) Uid:(504/lee) Gid:(502/lee)
Device:  3,0   Inode: 200705    Links: 30
Access: Thu May 27 10:54:55 1999(00000.00:01:43)
Modify: Thu May 27 10:44:41 1999(00000.00:11:57)
Change: Thu May 27 10:44:41 1999(00000.00:11:57)
```

5.13 Links

- More than one filename can refer to an inode
  - ◇ These file names are *links* to the file
- `ln` creates links to files
  - ◇ Creates *hard links* by default
  - ◇ `ln -s` creates *symbolic* or *soft links*
- Erasing a file just removes its directory entry
  - ◇ The file is only lost when all entries for it have been removed
- Crucially : A filename is *not* the file
  - ◇ The inode *is* the file
  - ◇ All names are simply links (references) to the inode
  - ◇ Vague resemblance to Windows' 'shortcuts'

5.14 Hard links

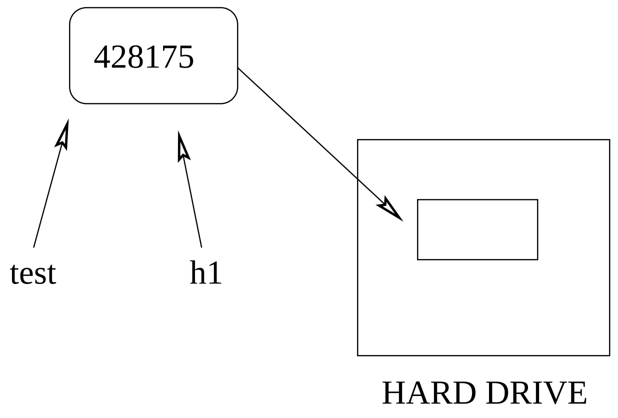
- A *hard link* is merely a directory entry with the relevant *inode* number
- Consider the following
  - ♢ Start with:

    ```
    $ ls -li
    428175 -rw-rw-r-- ... 4 May 26 13:18 test
    ```

  - ♢ We create a *hard link*:

    ```
    $ ln test hl
    $ ls -li
    428175 -rw-rw-r-- ... 4 May 26 13:18 hl
    428175 -rw-rw-r-- ... 4 May 26 13:18 test
    ```

- N.B. *Hard links* cannot cross filesystems
- *Inode* numbers are filesystem specific



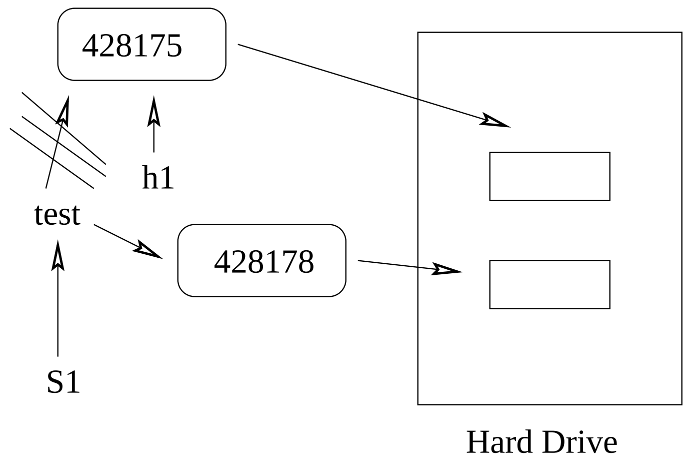
5.15 Soft links

- *Soft links* store the pathname of the linked file
- This means they can cross filesystems
- Adding a *soft link* :

```
$ ln -s test sl
$ ls -li
428175 -rw-rw-r-- ... 4 May 26 13:18 hl
428176 lrwxrwxrwx ... 4 May 26 13:19 sl -> test
428175 -rw-rw-r-- ... 4 May 26 13:18 test
```

- If we replace the `test` file with another then the symbolic link still works, but the hard one still points to the old file!

```
$ mv ../test2 test
$ ls -li
428175 -rw-rw-r-- ...  5 May 26 13:45 hl
428176 lrwxrwxrwx ...  4 May 26 13:19 sl -> test
428178 -rw-rw-r-- ... 15 May 26 13:47 test
```



5.16 Access Control and UID

- File access can be limited to specific users
- *Super* user(s) can override access control
- Access control is set by user and group ID
- Each user has a user-id (*UID*) and one or more group-ids(*GIDs*)
- Processes have an associated UID and GID
  - ♢ Inherited from the user who created the process
- They can however can be changed:
  - ♢ Processes are known as set-user ID (setuid) if they set their own user ID
  - ♢ or set-group ID (setgid) if they set their own group ID

5.17 Categories of Access Control

- There are three categories of access

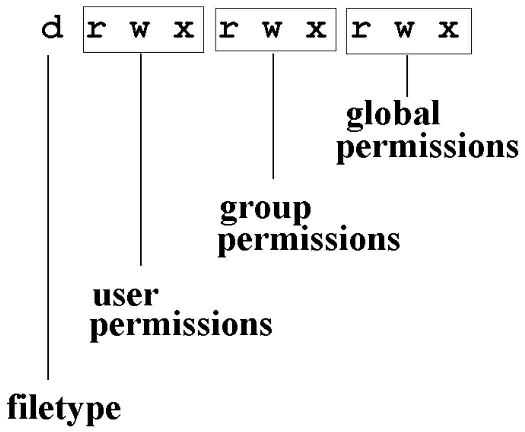
read	`r`
write	`w`
execute	`x`

- These may be specified for three sets of users:
  - ♢ User
  - ♢ Group
  - ♢ Everyone

5.18 Access Control - Example

- `ls -l` shows the access permissions, e.g.

```
$ ls -l
-rw-rw-r-- 1 www    www    x_windows.tex
lrwxrwxrwx 1 lee    lee    img -> ../linux/img/
-rw-rw-r-- 1 lee    lee    test.log
```



5.19 Changing Access Permission: `chmod`

- Only the owner of a file (or the super-user) may alter its access permissions
- `chmod` (change mode) changes access permissions
  - ♢ Works in two ways, symbolically or numerically
  - ♢ Symbolically is easier to remember (for most)

5.20 `chmod` symbolically

- Select who you want to change permissions for `(u=user, g=group, o=others, a=all)`
- Decide whether you want to `grant a permission(+), remove(-), or set(=)` it
- Take the permission that you want to change `(r=read, w=write, x=execute)`
- Example :

  ```
  $ chmod gu+w filename
  ```

  Adds write permission for user and group
- You can make several changes by separating the settings with commas, e.g.

  ```
  $ chmod a-w,gu=rw filename
  ```

  Removes write permission for all, then grants it for the user and group

5.21 `chmod` numerically

- Once you know this it is often quicker
- A number represents each permission type

4	read permission
2	write permission
1	execute permission

- Add up the permission numbers you want for each user group (owner, group, all) and supply these to `chmod`
- Example:

```
$ chmod 755 filename
```

grants all permissions to the owner (4+2+1), and read and execute (4+1) to group and all others

5.22 `umask`

- Files begin with a default access setting
  - ♢ Specified by a user's *umask* setting
- This only works numerically
- Unlike `chmod`, specified permissions are turned *off*
- With a umask setting of 000 files are created with permissions `rw-rw-rw-` (666)
- Default umask is 022 which means files are typically created `rw-r-r-` (644) [3] e.g.

```
$ umask
002
$ touch foo
$ ls -l foo
-rw-rw-r-- 1 lee  lee     0 Feb 10 17:17 foo
$ umask 222
$ touch foo2
$ ls -l foo2
-r--r--r-- 1 lee  lee     0 Feb 10 17:17 foo2
```

[3]This is the case on Redhat systems where users typically belong to a group of their own, other distributions will probably use a default umask of 022

5.23 Special Files - `/dev`

- Files under `/dev` typically represent devices attached to your computer
- Programs can open and close them and read from and write to them - as with regular files
- Kernel code handles exactly how these work
- Two types
  - ♢ Block - Disk drives, tape drives, CDROMs
  - ♢ Character - Printers, modems, etc.

5.24 Special Files - `/proc`

- The section of the filesystem called `/proc` doesn't contain *real* files
- It contains system status information
- For example:

Location	Information
/proc/[number]	On specific running processes. See `man proc` for details
/proc/meminfo	How much memory is in your system and how much is being used
/proc/cpuinfo	What CPU(s) you are currently using
/proc/filesystems	Filesystems your kernel supports
/proc/kcore	An image of your physical memory
/proc/net	Network status of your machine
/proc/pci	PCI devices found at initialization
/proc/sys	Details on kernel variables, e.g. ◇ Maximum number of files we can open (file-max) ◇ Number of files currently open (file-nr) ◇ The uptime of the system (uptime)

Table 5.1: System Information from `/proc`

5.25 Filesystem Structure

Multi-Volume Filesystems

- The filesystem can be held on several devices
- Large disks can be divided into partitions
  - ◇ This creates several *logical* devices
- A basic Linux system must be present on `/`
- Other parts of the fs may be mounted at any time
- The main ones are mounted at boot time
- This is controlled by the `/etc/fstab` file which says which volumes are mounted where

5.26 `/etc/fstab` - Example

Logical Volume	Mount Point	FS type	Options	Dump	Check order
/dev/hda1	/	ext2	defaults	1	1
/dev/hda5	/home	ext2	defaults	1	2
/dev/hda7	/tmp	ext2	defaults	1	2
/dev/hda6	/usr	ext2	defaults	1	2
/dev/hda8	swap	swap	defaults	0	0
/dev/fd0	/mnt/floppy	ext2	noauto	0	0
/dev/cdrom	/mnt/cdrom	iso9660	noauto,ro	0	0
\\kashmir\c	/mnt/kashmir	smbfs	guest	0	0
landlord:/var/admin	/var/admin	nfs	defaults	0	0
landlord:/home/lee	/home/lee/LANDLORD	nfs	defaults	0	0

5.27 Mounting Additional Volumes

- To mount a filesystem use `mount`, e.g.

  ```
  mount /dev/cdrom /mnt/cdrom
  ```

- Mounts the filesystem `/dev/cdrom` in the directory `/mnt/cdrom`
- `cd /mnt/cdrom` changes directory to the root of the CDROM's filesystem
- To unmount use `umount name` where *name* is either the filesystem name or the mount point:

  ```
  umount /dev/cdrom
  umount /mnt/cdrom
  ```

- *N.B.* - A filesystem can only be unmounted when it is no longer in use. *'In use'* includes :
  - ◊ Having any file on that filesystem open
  - ◊ Having a shell in a directory on that filesystem

5.28 Mounting shared filesystems

- *NFS* filesystems can be mounted with

```
mount -t nfs hostname:path mount-point
```

- Example:

```
mount -t nfs landlord:/backup /mnt/backup
```

- Share files from MS-Windows machines using *SAMBA*
- This is a free implementation of the Windows file-sharing protocols, e.g.

```
mount -t smbfs '\\ntbox\c' /mnt/ntbox
```

- *N.B.* Linux does not use the ‘drive letter’ concept at all
  - ◊ *Drives* and *shares* integrate seamlessly into the filename tree

5.29 Summary

- The primary Linux filesystem is Ext2
- It has a tree-like hierarchy of directories
- Directories merely contain pointers to files (*inodes*)
- *inodes* contain all the information about a file
- Can have multiple links to the same file
- Read/Write access is controlled per file
- Creation/Deletion of files is controlled by permissions of the directory
- Several filesystems can be mounted to create the directory hierarchy

5.30 Filesystem Exercises

1. *Basic navigation*

   (a) Log in and use `pwd` to discover what the full path of your home directory is.

   (b) Change directory to `/bin` and then `/tmp`. Use `pwd` to check you got there each time.

   (c) When in `/tmp` type `cd ..` and use `pwd` to find out where you end up.

   (d) What is the parent directory of the root of the filesystem? Why is this so?

   (e) Move back to your home directory. Think of three ways you can do this.

2. *Directories*

   (a) Start in your home directory and create a directory called `new`

   (b) Change to the `new` directory and create a directory called `newer`

   (c) Go to you home directory. Now create a directory under `newer` called `newerstill` There are two ways to do this what are they? (Hint: You don't *have* to change directories to solve this.)

   (d) Remove all the directories that you've just created, there are several ways to do this.

   (e) Create the same directory structure with one command.

3. *Links*

   (a) Create a file called `test` in your home directory (Typing `echo 123> test` should do this). Now create a hard link to `test` called `h_test` and a symbolic link to test called `s_test`

   (b) Find out the inode number of the files. Check you understand why they are what they are.

   (c) Remove the original file called `test`. Can you still get at the contents of the original file?

   (d) What happens if you try `cat s_test`. Make sure you understand the distinction between `h_test`, and `s_test`

   (e) Try to make a *hard link* to your home directory. Why does this fail?

4. `/proc`

   (a) Use the files in `/proc` to find out how much memory your system has and what processor it is running on.

   (b) Find out what PCI devices are attached to your machine.

   (c) Find out what environment variables are set for your currently running shell using the information in `/proc`. *Hint* you can get the process-id of your shell using $$

   (d) Whether or not your machine is doing IP forwarding is stored in the file `/proc/sys/net/ipv4/ip_forward`. You can `cat` this file, a value of 1 means that IP forwarding is turned on. Find out whether or not your machine will forward IP.

   (e) Find out how many files are currently open on your system.

5.31 Filesystem Solutions

1. *Basic Navigation*

   (a) You will probably see

   ```
   $ pwd
   /home/username
   ```

   where `username` is the name you log in with.

   (b) Check that the output of `pwd` is `/bin` and `/tmp`

   (c) You should end up in `/` This is the root of the filesystem.

   (d) The parent of `/` is itself. You can use `ls -lia` to show that the inode numbers for `.` and `..` are the same when you are in `/`

   (e) You can move back to your home directory by using any of the following:

   i. `cd`

   ii. `cd ~`

   iii. `cd /home/username` (Provided your home directory lives under /home)

2. *Directories*

   (a)
   ```
   $ cd
   $ mkdir new
   ```

   (b)
   ```
   $ cd new
   $ mkdir newer
   ```

   (c)
   ```
   $ cd
   ```
   then either :

   i.
   ```
   $ cd new/newer
   $ mkdir newerstill
   ```

   ii.
   ```
   $ mkdir new/newer/newerstill
   ```

   (d) Any of the following will work :

   i.
   ```
   $ cd
   $ cd new/newer
   $ rmdir newerstill
   $ cd ..
   $ rmdir newer
   $ cd ..
   $ rmdir new
   ```

   ii.
   ```
   $ cd
   $ rmdir -p new/newer/newerstill
   ```

   iii.
   ```
   $ cd
   $ rm -fr new/
   ```

   (e)
   ```
   $ cd
   $ mkdir -p new/newer/newerstill
   ```

3. *Links*

   (a) This is achieved as follows:

   ```
   $ ln test h_test
   $ ln -s test s_test
   ```

(b) `ls -li` should show that the inode number of the original file and `h_test` are identical `h_test` is another name for the original file. The inode number for `s_test` will be different. It is a separate file that contains information about the location of the file it is a link to.

(c) After you created the hard link the original file had two names `test` and `h_test`. You have removed `test` but until all names for a file have been removed it is still accessible. In this case you can do `cat h_test` to see the contents of the file.

(d) This should fail with a `No such file or directory` message. `s_test` contained a pointer to the file `test` not the inode number. There is no longer a file named `test` so this cannot work. *Hard links* reference a file by its inode number, *symbolic links* reference it by its name

(e) This is not allowed, as it could stop the filesystem being strictly hierarchical.

4. `/proc`

(a) `cat /proc/meminfo` should show you memory usage.

(b) `cat /proc/pci` gives a list of all PCI devices.

(c) `cat /proc/$$/environ` will give a list of the environment of your current shell. Each variable is delimited with the nul character (decimal 0). The following will show the output with one variable per line :

```
$ cat /proc/$$/environ | tr '\verb|\|000' '\verb|\|n'
```

(d) See The Exercise -

(e) `cat /proc/sys/fs/file-nr`

Module 6

More on the Filesystem

Objectives

After completing this module you should be able to understand and utilise these features of the Linux filesystem:

- Inodes
- Non-native filesystem types
- Filesystem checking and recovery
- Disk usage utilities
- Disk partitioning utilities
- Filesystem creation
- Block, Character and ‘Raw’ Devices

6.1 Inodes in Depth

- Every file/directory is specified by an *inode* [1]
- A single inode can have many filenames (links)
- An *inode* contains information about:
  - ♢ File type (ordinary, directory, FIFO, block device etc.)
  - ♢ Owner ID (user the file belongs to)
  - ♢ Size (in bytes)
  - ♢ Access, creation, and modification times
  - ♢ Group ID (to which group does the file belong)
  - ♢ File permissions
  - ♢ Mapping of the file contents (data sectors)
- Knowing inode numbers can be very useful if you want to restore files on a disk with bad blocks
- Inode layout and location varies with fs type

[1] The term *inode* was invented by Dennis Ritchie of AT&T. He admits to forgetting why he chose that name.

6.2 Inodes (*continued*)

- `ls -i` displays inode numbers of filenames, e.g.

```
200808 Ext2fs-0.1-14.html   542726 include
200795 Ext2fs-0.1-2.html    188447 info
200797 Ext2fs-0.1-4.html    333831 ldap
200802 Ext2fs-0.1-8.html    329729 man
200803 Ext2fs-0.1-9.html    278533 misc
200793 Ext2fs-0.1.html      428042 nsmail
204802 systemprogramming
```

- `stat` prints the inode contents for files inc. permissions, size, links, access times etc.

```
$ stat /home/lee
File: "/home/lee"
Size: 4096         Filetype: Directory
Mode: (0755/drwxr-xr-x) Uid:(504/lee) Gid:(502/lee)
Device:  3,0   Inode: 200705    Links: 30
Access: Thu May 27 10:54:55 1999(00000.00:01:43)
Modify: Thu May 27 10:44:41 1999(00000.00:11:57)
Change: Thu May 27 10:44:41 1999(00000.00:11:57)
```

6.3 Links

- More than one filename may refer to an inode
  - ◇ These file names are *links* to the file
- `ln` creates links to files
  - ◇ Creates *hard links* by default
  - ◇ `ln -s` creates *symbolic* or *soft links*
- Erasing a file simply removes its directory entry
  - ◇ Only when all entries for a file have been removed is the file lost
- Crucially : A filename is *not* the file
  - ◇ The inode *is* the file
  - ◇ All names are simply links (references) to the inode
  - ◇ Vague resemblance to Windows' 'shotcuts'

6.4 Hard links

- A *hard link* is merely a directory entry with the relevant *inode* number.
- Consider the following
  - ◇ We start with :

    ```
    $ ls -li
    428175 -rw-rw-r-- ... 4 May 26 13:18 test
    ```

  - ◇ We create a *hard link* :

    ```
    $ ln test hl
    $ ls -li
    428175 -rw-rw-r-- ... 4 May 26 13:18 hl
    428175 -rw-rw-r-- ... 4 May 26 13:18 test
    ```

- N.B. *Hard links* cannot cross filesystems
- *Inode* numbers are filesystem specific

6.5 Soft links

- *Soft links* store the pathname of the linked file
- This means they can cross filesystems
- Adding a *soft link* :

```
$ ln -s test sl
$ ls -li
428175 -rw-rw-r-- ... 4 May 26 13:18 hl
428176 lrwxrwxrwx ... 4 May 26 13:19 sl -> test
428175 -rw-rw-r-- ... 4 May 26 13:18 test
```

- If we replace the `test` file with another then the symbolic link still works, the hard one still points to the old file!

```
$ mv ../test2 test
$ ls -li
428175 -rw-rw-r-- ...  5 May 26 13:45 hl
428176 lrwxrwxrwx ...  4 May 26 13:19 sl -> test
428178 -rw-rw-r-- ... 15 May 26 13:47 test
```

6.6 Non-native Filesystems

- Besides Ext2, Linux supports most well-known filesystems, e.g.
  - ♢ MS-DOS (FAT16), VFAT, FAT32
  - ♢ ISO9660 (CD-ROM)
  - ♢ NTFS (Windows NT)
  - ♢ SMB / CIFS (MS Windows file sharing)
- Most can be mounted, read & written
- Specific tools can *create* some non-native filesystems (e.g. DOS/CD-ROM)
- Some non-native filesystems can even be checked and repaired (e.g. Minix)
- *Not* wise to automate checking and repair on non-native systems

6.7 Disk Checking and Recovery (`fsck`)

- `fsck` checks and repairs Linux filesystems
- GENERIC SYNTAX:

```
$ fsck [general options] [-t fstype] [fs-options]
```

- Actually, just a front-end for specific filesystem checkers , e.g. `e2fsck`
- The filesystem-specific checker is searched for in these places in this order:
  - ♢ `/sbin`
  - ♢ `/etc/fs` and `/etc`
  - ♢ Directories in the PATH environment variable
- Common options:

  `-A` uses `/etc/fstab` to ID and check *all* filesystems in one run

  `-R` skips the root filesystem when the `-A` option is operative

6.8 Check a Linux Ext2 filesystem (`e2fsck`)

- `e2fsck` is the application called by `fsck` to check ext2 filesystems
- SYNTAX
  `$ e2fsck [options] device`
- Devices specified as `/dev/hdXX`, `/dev/sdXX`, etc
- Main options:

Option	Effect
`-c`	`e2fsck` finds bad blocks and marks them by adding them to the *bad block inode*
`-f`	Force checking even if the file system seems clean
`-n`	Usually opens filesystem read-only and answers “no” to all questions Note: if the `-c, -l, or -L` options are also specified the filesystem will be opened read-write, to update bad-blocks list, but no other changes are made
`-p`	Automatically repair the filesystem without questions
`-v`	Verbose mode
`-y`	Answer of “yes” to all questions

6.9 Disk Free Space (`df`)

- `df` shows free space on filesystems
- SYNTAX:

  `$ df [`*`options`*`]` [*filesystem list*]
- Defaults to *all* currently mounted filesystems
- Displays in 1K blocks by default; environment variable POSIXLY_CORRECT sets 512-byte
- GNU `df` can't show space on unmounted FSs
- Main options:

Option	Effect
`-s`	Include *all*, including pseudo-filesystems
`-i`	List inode usage instead of block usage
`-k`	1K block output. Override POSIXLY_CORRECT
`-P`	POSIX output, i.e. every fs on a single line. Columns are often misaligned
`-T`	Display fstypes in output
`-t`	Limit output to specific fstypes. Multiple -t options poss
`-x`	Exclude specific fstypes
`-h`	'Human-readable' output

6.10 Disk Usage (`du`)

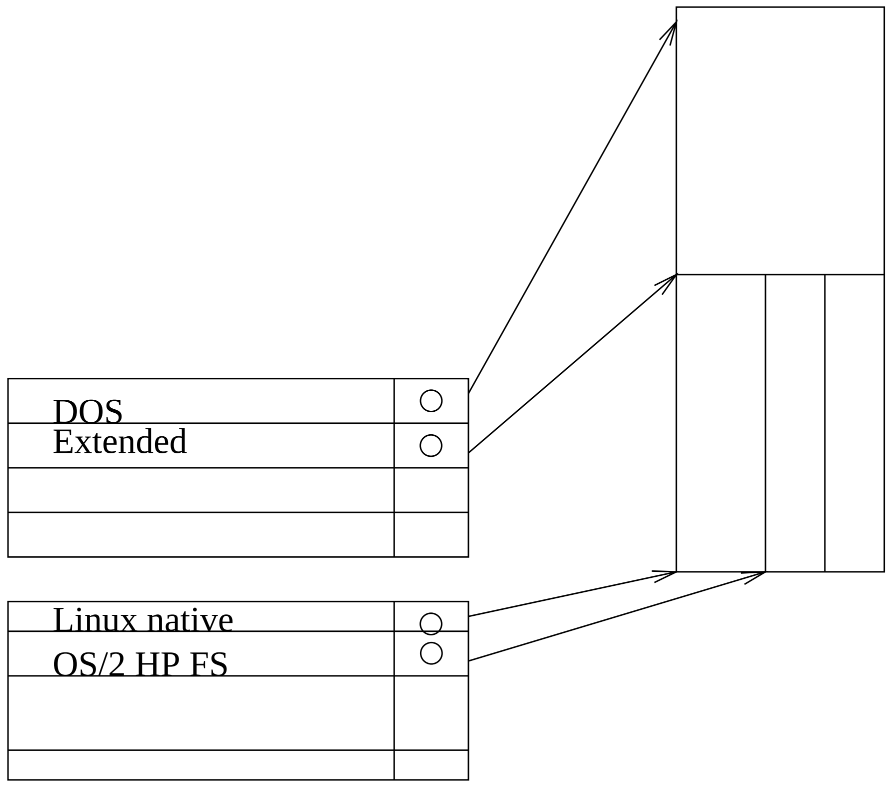
- `du` reports disk space used by a directory (inc. its sub-directories) or a file
- Useful for summarizing file/directory sizes
- SYNTAX:
  `$ du [`*options*`] [`*path list*`]`
- GNU `du` defaults to 1K blocks, unless set POSIXLY_CORRECT (512-byte)

Option	Effect
`-a`	Display counts for *all* files, not just directories
`-c`	Cumulative totals after all arguments have been processed. Useful to calculate directory usage, with some files excluded
`-k`	Output in Kbs. Overrides POSIXLY_CORRECT
`-s`	Displays only totals for directory/file; ignoring sub-directories
`-h`	'Human-readable' output

6.11 Disk Partitioning Concepts

- Partitioning enables efficient use of large drives by dividing them into smaller sections
- A *Partition table* at the start of each disk points to the beginning/end of each partition
- Max 4 *Primary Partitions* on standard disks
- More possible *inside Extended Partitions*
- An Extended Partition has its own partition table; pointing to sub-divisions *within* it
- Sub-divisions called *Logical partitions* (drives)

6.12 A Partitioned Disk



6.13 Making and Changing Partitions

- Linux usually benefits from putting parts of its filesystem on different drives or partitions

  e.g. access different parts simultaneously

- Issues:
  - ◇ How many partitions?
  - ◇ Size of partitions for specific directories
  - ◇ Partitioning an empty drive
  - ◇ Destructive re-partitioning
  - ◇ Non-destructive re-partitioning

6.14 How Many Partitions?

- How many partitions you need depends on:
  - ♢ How you intend to use a system
  - ♢ What resources you have (e.g. physical disks)
- A common single-disk server might have separate partitions for:
  - ♢ Swapping
  - ♢ The `/` filesystem/directory
  - ♢ The `/boot` filesystem/directory
  - ♢ The `/home` filesystem/directory
  - ♢ The `/usr` filesystem/directory
  - ♢ The `/var` filesystem/directory
  - ♢ The `/usr/doc` filesystem/directory

6.15 What Size Partitions?

- What size partitions you need depends entirely upon how you intend to use the system
- Filesystems needing large partitions are usually:

  `/home` for users' file space
  `/var` for server files (news, web, mail, logs etc)

- The following are usually made just big enough:
  - ♢ the swap partition (rarely more than 127 Mb)
  - ♢ `/boot` (enough for a few alternative kernels)

6.16 BIOS Problems With LILO and Partitions

- BIOSes on most Intel-type machines can't access data beyond cylinder 1023
- To start Linux, LILO uses data in `/boot`
- The `/boot` partition *must always* be located *entirely* below cylinder 1023
- Multiple IDE drive machines always work if `/boot` is on the *first primary controller*
- On mixed IDE/SCSI systems `/boot` *must* be on the *first* IDE primary controller or SCSI `ID 0`
- On multiple SCSI drive systems `/boot` *must* be on `ID 0` or `ID 1`

6.17 Disk Partitioning Tools

- Common partitioning tools:

Tool	Notes
`fdisk`	Standard on all Linux and much UNIX. Very flexible. Tricky character-based interface. Fairly reliable
`fips`	Non-destructive re-sizing of partitions. Difficult CLI. Some distrust it.
`Disk Druid`	Red Hat install-time tool. Friendlier character-based interface. Set growable partitions. Set mount points
`Partition Magic Lite`	Used in Caldera X-based installation. Allows non-destructive repartitioning on the fly. Easy and reliable. Proprietory. Not always available
`Partition Magic`	As above, but the full commercial product and price

6.18 Using `fdisk`

- `fdisk` basically *writes* partition tables
- Very dangerous, but not *that* difficult
- SYNTAX:

  `$ fdisk [`*options*`] [`*disk-device-name*`]`
- One useful CLI option:

  `-l` lists available partition tables then exits
- Usually superuser-only and needs explicit path
- 2 tips:
  - ◇ *Never* use `w` (write/save changes) unless you really know what you are doing
  - ◇ Use `q` (quit) to exit without saving changes

6.19 The `fdisk` Interface

- The only helpful thing about `fdisk` screens is that they all prompt with (`m` for help)
- Launched without arguments on IDE:

```
[root@oakleigh davef]# /sbin/fdisk
Using /dev/hda as default device!

Command (m for help):
```

- `p` prints the partition table for current hard disk

```
Disk /dev/hda: 64 heads, 63 sectors, 524 cylinders
Units = cylinders of 4032 * 512 bytes

  Device Boot Begin Start End Blocks  Id  System
/dev/hda1 *     1     1  204 411232+  6  DOS 16-bit >=32M
/dev/hda2     205   205  524 645120   5  Extended
/dev/hda5     205   205  459 514048+ 83  Linux native
/dev/hda6     460   460  523 128992+ 82  Linux swap
```

6.20 Interactive Commands in `fdisk`

- A selection of `fdisk` interactive commands:

Tool	Notes
`a`	toggle partition as boot*able*
`d`	*Delete* a partition. Needed before re-allocating disk space to other partitions
`l`	*List* known partition types
`m`	Help *menu*
`n`	Create/add a *new* partition
`p`	*Print* to screen current drive partition table
`q`	*Quit* without saving changes
`t`	Change a partition’s filesystem *type*
`w`	*Write* table to disk and exit
`x`	*Experts* (i.e. total gurus) only

6.21 Making Linux Filesystems (`mke2fs`)

- Normally, you can only write a file to a device with a filesystem on it (formatted in DOS-speak) [2]
- Native Linux filesystems are made using `mke2fs`
- Utilities for other types: `mkdosfs`, `mkisofs`, etc
- SYNTAX:

```
$ mke2fs [options] device
```

- Filesystems are made on **un**mounted devices
- If omitted, `mke2fs` auto-calculates fs size
- Lots of options, few widely used:

Option	Effects
`-c`	Check for bad blocks before creating fs
`-i`	Specify the bytes/inode ratio. Defaults to 4096 bytes. Lower useful on floppies
`-m`	% of reserved blocks for super-user. Default 5%. 0% common on data floppies
`-q`	Quiet execution. Useful in scripts

[2] Some Linux utilities, like `tar` and `dd` can actually write to ‘raw’ devices

6.22 Block Devices

- Typically, *block devices* are disks and tapes
- Strictly, I/O devices with these characteristics:
  - ♢ Seen by kernel as a range of blocks 0 to *n*-1
    - ∗ Where *n* = number of blocks on the device
  - ♢ Can have a filesystem mounted on it
  - ♢ Ability to perform random access reads
  - ♢ A specific block size
  - ♢ Handling only one data block at a time
  - ♢ Only accepts actions on whole data blocks
  - ♢ Kernel buffers its I/O

6.23 Character Devices

- Any device which is *not* a block device
- Typically:
  - ♢ printers
  - ♢ terminals
  - ♢ modems
- Drivers determine how a program reads from and writes to it
- For example, a terminal device driver lets programs read typed info in two ways:
  - ♢ In *raw* mode (i.e without driver interpretation)
  - ♢ A line at a time with the driver removing erase and kill chars (typos and corrections), so:
    - ∗ the program reads everything on a line
    - ∗ the number of characters on a line can vary
- Kernel doesn't buffer character devices

6.24 More Filesystem Exercises

1. *Inodes and Linking*
   (a) Use `stat` to find out the number of hard links to your `/` directory.
   (b) Use the following to locate the files with inode numbers 1 and 3:
   ```
   $ find / -inum inode-number
   ```
   (c) Create a file called `test` in your home directory (Typing `echo > test` should do this). Now create a hard link to `test` called `h_test` and a symbolic link to test called `s_test`
   (d) Find out the inode number of the files. Check you understand why they are what they are.
   (e) Remove the original file called `test`. Can you still get at the contents of the original file?
   (f) What happens if you try `cat s_test`. Make sure you understand the destinction between `h_test`, and `s_test`
   (g) Try to make a *hard link* to your home directory. Why does this fail?

2. *Creating Filesystems and Formatting*
   (a) Identify and *use* the command string needed to create the following on a floppy disk:
      i. An Ext2 filesystem
      ii. A DOS filesystem
      iii. An IS09660 CD-ROM filesystem
   (b) To test your DOS filesystem, `cp` a long filename file to it. What happens to the filename?
   (c) Mount the ISO9660 formatted disk under `/mnt/floppy` to prove it is readable.
   (d) Identify and *use* the command string needed to do the following to a floppy disk:
      i. Low-level format the disk
      ii. Add a DOS fs to a low-level formatted disk
   (e) Try using `tar` to write to a floppy disk *without* a filesystem on it (you may have to low-level format the disk to remove an existing fs).

3. *Checking and Repairing Filesystems*
   (a) Find the appropriate man pages and, hence, the commands to:
      i. Locate bad blocks on an Ext2 device
      ii. Find and mark the bad blocks on DOS/FAT floppies
   (b) Use `e2fsck` to force checking and automatically repair an Ext2 formatted floppy.

4. *Creating, Modifying and Deleting Partitions*

   It is *not* wise to practice disk partitioning on important filesystems. It can be done, but mistakes may be unrecoverable. So we ask you to do the following partitioning questions on a floppy. Most people won't be able to make mountable filesystems on a floppy (you need to make a block device file first), but you should get some safe practice on `fdisk`. You may only be able to do these floppy-based questions using recent versions of `fdisk`.

   (a) Delete all existing partitions from your floppy

(b) Create 2 new primary partitions:
   i. 1 with an Ext2 system ID
   ii. 1 with a Win95 FAT32 system ID

(c) Save the new partition table and exit `fdisk`

(d) Re-enter `fdisk` and check the floppy's partition table contains the correct partitions

(e) Delete the FAT32 partition and replace it with two FAT32 *logical* drives.

6.25 More Filesytem Solutions

1. *Inodes and Linking*

   (a) The answer depends on your system. In recent Red Hat distributions `/` is inode number 2.

   (b) The answer depends on your system. In recent Red Hat distributions inode number 1 is `/proc` and number 3 is `/proc/uptime`

   (c) This is achieved as follows:

   ```
   $ ln test h_test
   $ ln -s test s_test
   ```

   (d) `ls -li` should show that the inode number of the original file and `h_test` are identical `h_test` is another name for the original file. The inode number for `s_test` will be different. It is a seperate file that contains information about the location of the file it is a link to

   (e) After you created the hard link the original file had two names `test` and `h_test`. You have removed `test` but until all names for a file have been removed it is still accessible. In this case you can do `cat h_test` to see the contents of the file

   (f) This should fail with a `No such file or directory` message. `s_test` contained a pointer to the file `test` not the inode number. There is no longer a file named `test` so this cannot work. *Hard links* reference a file by it's inode number, *symbolic links* reference it by its name

   (g) This is not allowed, as it could stop the filesystem being strictly hierarchical

2. *Creating Filesystems and Formatting*

   (a) These are typical solutions, but there are others:

      i. `$ mke2fs /dev/fd0`
      ii. `$ mkdosfs /dev/fd0`
      iii. `$ mkisofs -o /dev/fd0` *`source-filename`*

   (b) The filename should be shortened to 8 chars, without warning ... could be tricky!

   (c) Yes, you can mount a floppy drive on Linux with a CD-ROM format. Indeed, you can actually mount an ISO9660 formatted *disk image* (i.e. a single file) of an entire Linux directory tree. Trust me, it isn't as daft as it sounds!

   (d) Something like this:

      i. `$ fdformat /dev/fd0H1440`
      ii. Add a DOS fs to a low-level formatted disk

   (e) Use a command something like this:
   `$ tar -c` *`filenames`* `> /dev/fd0`

3. *Checking and Repairing Filesystems*

   (a) `apropos` should yield:

      i. `badblocks (8)`
      ii. `mbadblocks (1)`

   (b) `e2fsck -p -f /dev/fd0`
   Assuming that your floppy is actually OK, you will hardly get any feedback on stdout

4. *Creating, Modifying and Deleting Partitions*

   You probably can't do these floppy-based questions on old versions of `fdisk`.

   (a) Use `$ fdisk /dev/fd0` followed by the `d` interactive command
   (b) Use the `n` interactive command twice, followed by:
      i.
      ```
      Command (m for help): t
      Partition number (1-4): 1
      Hex code (type L to list codes): 83
      ```
      ii.
      ```
      Command (m for help): t
      Partition number (1-4): 1
      Hex code (type L to list codes): b
      ```
   (c) Use the `w` interactive command followed by the `q` interactive command
   (d) `$ fdisk /dev/fd0` followed by the `p` interactive command
   (e) Your screen should look something like this:

```
Command (m for help): p

Disk /dev/fd0: 2 heads, 18 sectors, 80 cylinders
Units = cylinders of 36 * 512 bytes

    Device Boot    Start       End    Blocks   Id  System
/dev/fd0p1             1        40       711   83  Linux
/dev/fd0p2            41        80       720    b  Win95 FAT32

Command (m for help): d
Partition number (1-4): 2

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
e
Partition number (1-4): 2
First cylinder (41-80, default 41):
Using default value 41
Last cylinder or +size or +sizeM or +sizeK (41-80, default 80):
Using default value 80

Command (m for help): p

Disk /dev/fd0: 2 heads, 18 sectors, 80 cylinders
Units = cylinders of 36 * 512 bytes

    Device Boot    Start       End    Blocks   Id  System
/dev/fd0p1             1        40       711   83  Linux
/dev/fd0p2            41        80       720    5  Extended

Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (41-80, default 41):
```

```
Using default value 41
Last cylinder or +size or +sizeM or +sizeK (41-80, default 80): 60

Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (61-80, default 61):
Using default value 61
Last cylinder or +size or +sizeM or +sizeK (61-80, default 80):
Using default value 80

Command (m for help): p

Disk /dev/fd0: 2 heads, 18 sectors, 80 cylinders
Units = cylinders of 36 * 512 bytes

    Device Boot    Start       End    Blocks   Id  System
/dev/fd0p1             1        40       711   83  Linux
/dev/fd0p2            41        80       720    5  Extended
/dev/fd0p5            41        60       351   83  Linux
/dev/fd0p6            61        80       351   83  Linux

Command (m for help): t
Partition number (1-6): 5
Hex code (type L to list codes): b
Changed system type of partition 5 to b (Win95 FAT32)

Command (m for help): t
Partition number (1-6): 6
Hex code (type L to list codes): b
Changed system type of partition 6 to b (Win95 FAT32)

Command (m for help): p

Disk /dev/fd0: 2 heads, 18 sectors, 80 cylinders
Units = cylinders of 36 * 512 bytes

    Device Boot    Start       End    Blocks   Id  System
/dev/fd0p1             1        40       711   83  Linux
/dev/fd0p2            41        80       720    5  Extended
/dev/fd0p5            41        60       351    b  Win95 FAT32
/dev/fd0p6            61        80       351    b  Win95 FAT32
```

Module 7

Shared File Systems

Objectives

After completing this chapter you should be able to:

- Understand basic remote file and print sharing
- Appreciate the pros and cons of Samba and NFS
- Install Samba and NFS servers
- Configure basic Samba and NFS services
- Access remote resources using SMB and NFS

7.1 NFS (Network File System)

- NFS developed by Sun Microsystems (early 80’s)
- Native method for file sharing between Unix/Linux systems
- Stateless protocol
  - ♢ Means server keeps no state
  - ♢ Renders server crashes ‘easily recoverable’
- Should be compatible with all Unix-like systems
- Best in trusted environment, not highly secure
- Best where all user/group IDs are same
- Often used with *Network Information Services* (NIS) to synchronise user/group IDs

7.2 NFS Basics . . . continued

- Systems are clients, servers or both
- Clients *import* shared filesystems
- Servers *export* shared filesystems
- Servers easy to implement via network daemons
- Clients require kernel modifications
- Linux systems normally work as both already
- NFS is NOT Unix/Linux specific (e.g. PC-NFS)

7.3 Exporting File Systems

- Exporting handled by daemons `rpc.nfsd` and `mountd`
- Must be running for NFS export to work
- Exported file systems listed in `/etc/exports`, format is:

  ```
  fsname hostname(flags) [hostname(flags)]
  ```

- Example:

  ```
  /tmp *.blah.co.uk(ro)
  ```

  Exports `/tmp` to all systems belonging to domain read-only [1]

- Important flags:
  - ♢ `ro` (read only)
  - ♢ `rw` (read/write)
  - ♢ `all_squash` (map all uid/gid to something)
  - ♢ `anonuid` (specify user ID to map to)
  - ♢ `anongid` (specify group ID to map to)
- After changing `/etc/exports`, restart NFS

  ```
  killall -HUP rpc.nfsd
  killall -HUP mount
  ```

  or

  ```
  /etc/rc.d/init.d/nfs restart
  ```

[1]For full detail on flags use `man exports`

7.4 Viewing exports

- Use showmount:

```
$ showmount -e
$ showmount -e hostname
```

```
Export list for landlord.gbdirect.co.uk:
/usr/local/gbdirect/cvsroot roti.gbdirect.co.uk
/home/adamg                 roti.gbdirect.co.uk
/home/andylong              along2.gbdirect.co.uk
/home/mikeb                 kebab.gbdirect.co.uk
/mnt/cdrom                  <anon clnt>
```

- `NFS` uses a `portmapper` to handle requests
- This must be running (and you must have access to it) to use `NFS`
- Check that `hosts.allow` contains an entry to permit you access, e.g.
  - ◊ `portmap:  ALL`
    or
  - ◊ `portmap:  my.ip.network.`

7.5 Importing File Systems

- Mount a remotely exported directory
- Usually have to be superuser

  ```
  $ mount hostname:/sharename /local/directory
  ```
- If successful, the export named `/sharename` on host *hostname* is mounted on your *mountpoint* `/local/directory`
- Files accessed just as if local
- Remote host must be exporting the directory
- You must have access permission
- Your local mountpoint must exist
- Exactly like mounting a device

7.6 Samba

- Implementation of *Server Message Block* protocol (SMB)
  - ♢ Core of Microsoft's file and print sharing
  - ♢ Now *'re-invented'* as *CIFS*
- Developed in Australia by Andrew Tridgell et al
- Info, sources, distributions at `www.samba.org`
- High performance – competitive with NT
- Server is purely application code
  - ♢ Not part of the OS
- Can by a client
  - ♢ Requires OS support
  - ♢ Client module (`smbfs`) not part of Samba

7.7 Samba Installation

- Will vary - may come preinstalled, may come as RPMs or similar
- Key components are `nmbd` and `smbd`
  - ♢ `nmbd` is the name services daemon; mostly fit-and-forget
  - ♢ `smbd` is the samba server; listens for connections and then forks one copy per client
- Other tools & utilities exist, e.g. `smbclient`
- Configuration file is `/etc/smb.conf`
- Later versions come with the *Samba Web Administration Tool* (`swat`); listens on port 901

7.8 Samba Basics

- Most likely started as *daemons* in *init scripts*
- Can be run-on-demand via `inetd`, but unlikely
  - ♢ Gives poor performance
- Exclusively uses *TCP/IP*. Microsoft clients need to be configured for it — they may use *NETBEUI*
- Permits:
  - ♢ full file sharing, browsing and domain controller services
  - ♢ full access to printers
  - ♢ extensive customising

7.9 Access to Files and Printers

- Linux and Win/NT access controls don’t match
- Various options can be set
- Attempts to match logged-on Windows Username to Linux user names and passwords
- Modern versions use encrypted passwords - takes some setting up (see documentation)
- Has concept of ‘guest’ users - may map to ‘nobody’ on Linux
- Take a look in your `smb.conf` file and read `man smb.conf`

7.10 Testing Samba

- Use `smbclient` (see screen dump below)
- May need to provide a password
- Check `DIAGNOSIS.TXT` from distribution (usually installed at `/usr/doc/samba`) if you have problems

```
$ smbclient -L localhost
Added interface ip=192.168.0.129 bcast=192.168.0.255
nmask=255.255.255.0
Password:
Domain=[GBDIRECT]     OS=[Unix]    Server=[Samba 2.0.3]

Sharename       Type    Comment
--------        ---     -------
www             Disk    WWW     files
software        Disk    Installable     Software
tmp             Disk    Temporary       file    space
admin           Disk    GBdirect        admin   files
printers        Printer All     Printers
IPC$            IPC     IPC     Service (Samba  Server)
okirmt          Printer
txtdj           Printer
djrmt           Printer
fax             Printer

Server          Comment
--------       -------
LANDLORD        Samba   Server

Workgroup       Master
--------       -------
GBDIRECT        LANDLORD
WORKGROUP       KEBAB
```

7.11 Smbclient

- Numerous options:

```
smbclient servicename  [password]  [-s
smb.conf]  [-B  IP addr]  [-O socket options][-
R name resolve order] [-M Net-BIOS name] [-i
scope] [-N] [-n  NetBIOS  name]  [-d  debu-
glevel] [-P] [-p port] [-l log basename] [-h]
[-I dest IP][-E] [-U username] [-L NetBIOS
name]  [-t  terminal  code][-m  max  protocol]
[-W workgroup] [-T<c|x>IXFqgbNan] [-Ddirectory]
[-c command string]
```

- Example:

```
$ smbclient //landlord/admin
Added interface ip=192.168.0.129
bcast=192.168.0.255 nmask=255.255.255.0
Password: xxxxx
Domain=[GBDIRECT] OS=[Unix] Server=[Samba 2.0.3]

smb: \> ls
  q3.dir                 85  Tue Jun 29 13:01:44 1999
  actwin2        D        0  Sun Mar  7 22:01:28 1999
  courses        D        0  Wed May 12 10:02:20 1999
  cvs            D        0  Mon Mar 22 12:36:13 1999
  domreg         D        0  Tue Sep  1 10:14:12 1998
  finance        D        0  Thu Jul  1 12:33:49 1999
  informat       D        0  Wed Jun 23 09:56:34 1999
  julie          D        0  Fri Jul  2 10:06:43 1999
............. etc ...........
```

7.12 Samba configuration File

- Three sections to `smb.conf`
  - ♢ global
  - ♢ directories
  - ♢ printers, if enabled, will export the printers known in `/etc/printcap`
- Far too much detail to go into here
- Lots of help in the HOWTO files
  - ♢ Usually under `/usr/doc/samba-versionnumber`, e.g. `/usr/doc/samba-2.0.5a`
- Read the man pages
- Via the web
- and others

7.13 Testing Samba

- Use `testparm` and `smbstatus`
  - ♢ `testparm` is used before starting Samba to check that `smb.conf` is ok
  - ♢ `smbstatus` reports status of Samba, all connected clients and file share modes

Notes on Testing Samba

- Note that Samba is a server implementation only
- Cannot be used by Linux to *import* shared files, only export them
- Some Linuxes have import facilities too — but requires kernel support (`smbfs` module)

7.14 Exercises

1. *NFS*
   (a) Set up your local host so you can use `showmount` to show exported directories.
   (b) Find other hosts on your network which list exports.
   (c) Set up your host to export `/tmp`
   (d) Go to some other system and mount the exported `/tmp`
   (e) Play with file access on the mountpoint!, e.g. Try accessing files you normally wouldn't have access to, creating files and seeing what the ownership and permissions are on the local copy.
2. *Samba*
   (a) Locate the file DIAGNOSIS.txt
   (b) Read through it, then carefully work through *all* of its instructions to check your Samba installation.
   (c) Run `testparm` on your current `smb.conf`, pipe the output through `less` to see the results.
   (d) Run `smbstatus` and explain to your neighbour what the results mean.
   (e) Set up a share so that your `/etc` directory is exported read-only and test it with `smbclient`.
   (f) Figure out how to export users' home directories and get a colleague to test your work.

7.15 Solutions

1. *NFS*

   (a) You should ensure that the `portmap` and `nfs` services are running before using `showmount`

   (b) You can give `showmount` a hostname to query, e.g.

   ```
   $ /usr/sbin/showmount -e somehost
   Export list for somehost:
   /home/adamg                 roti.gbdirect.co.uk
   /home/lee                   rafters.gbdirect.co.uk
   /backup                     <anon clnt>
   /home/james                 oakleigh.gbdirect.co.uk
   /mnt/cdrom                  <anon clnt>
   ```

   (c) You should add the following to `/etc/exports`:

   ```
   /tmp *(ro)
   ```

   (d) -

   (e) -

2. *Samba*

   (a) The file should be in `/usr/doc/samba-x.xx/docs/textdocs`

   (b) You should carry out all the test given to reach a working samba system

   (c) Check that your `smb.conf` is correct

   (d) Check the various smb manpages (`smb.conf`, `smbd`, `smbstatus` to see what the output means

   (e) You should add the following to your `smb.conf`, and restart samba with `/etc/rc.d/init.d/smb restart`:

   ```
   [etcshare]
   path = /etc
   comment = Shared etc directory
   writeable = no
           browseable = yes
   ```

(f) You should ensure that the *homes* share is uncommented in `smb.conf`, and restart samba if necessary. You can test this by using:

```
$ smbclient '\\localhost\username'
added interface ip=192.168.0.135 bcast=192.168.0.255 nmask=255.255.255.0
Password:
Domain=[MYGROUP] OS=[Unix] Server=[Samba 2.0.6]
smb: \> dir
.muttrc           H  1387  Thu Jan 27 11:45:21 2000
.addressbook.lu   H  2285  Mon Jan 24 14:37:29 2000
.procmailrc       H    38  Mon Jan 24 18:11:04 2000
.newsrc.eld       H  1151  Tue Jan 25 13:24:44 2000
.mail_aliases     H    56  Thu Jan 27 16:52:13 2000
Desktop           D     0  Tue Feb  8 15:48:40 2000
.opera           DH     0  Thu Jan 27 12:53:23 2000
.balsarc          H  1391  Wed Feb  9 14:27:36 2000
.mozilla         DH     0  Wed Feb  9 10:27:15 2000
ltculogo.gif         8202  Wed Feb  9 11:49:06 2000
LANDLORD          D     0  Wed Feb 16 14:49:16 2000
nltculogo.xcf       53886  Wed Feb  9 12:54:53 2000
nltculogo.gif        8398  Wed Feb  9 13:09:16 2000
```

Module 8

Firewalling and Network Security

Objectives

After completing this module you should be able to understand and utilise:

- Firewalling and Network Security principles
- Controlling access to daemons
- Basic firewalling with `ipchains`
- Network/routing debugging procedures
- Interface configuration under Linux
- The secure shell (`sshd`, `ssh`, and `scp`)
- NB: this is only an introduction. An in-depth treatment would take days.

8.1 Concepts

- Three important concepts:
  - ♢ Controlling network traffic into / through your system (packet filtering)
  - ♢ Controlling access to services / daemons
  - ♢ Avoid insecure services like `telnet`; replace with `ssh` etc.

8.2 What is Packet Filtering?

- Checks packet headers before acting on them
- Can ignore, reject or accept packets
- Makes decision based on source, destination, or packet type
  - ♢ Or a combination
- Set up using `ipchains` under kernel 2.2
- Older kernels used `ipfwadm`
- 2.4 kernel now uses `iptables` - possibly a bit early to trust
- More detailed treatment later

8.3 Controlling Access to Daemons

- Access control for run-on-demand daemons done with `inetd`
  - ◇ `/etc/hosts.allow`
  - ◇ `/etc/hosts.deny`
  - ◇ `/etc/inetd.conf`
- Flaw in `inetd` would still let things through
  - ◇ Best to drop the packets as soon as possible
  - ◇ So use packet filtering too

8.4 TCP Wrappers (`/usr/sbin/tcpd`)

- Raw `inetd` applies no access controls
- OK if you trust your network
- ‘TCP Wrappers’ invented to fix this
- Standard with most installations
  - ♢ the wrapper sits between `inetd` and the server daemon
- `inetd` not itself insecure
  - ♢ Insecurity springs from how you use it
  - ♢ Wrappers now integral with `xinetd`

8.5 TCP Wrapper Validation

- Uses `/etc/hosts.deny` & `/etc/hosts.allow`
- Well-documented, see `man hosts.allow`
- Example `/etc/hosts.deny`:
  `ALL: ALL`
  Denies all services to everyone
- Selectively enable trusted hosts in `/etc/hosts.allow`

```
in.telnetd:     .gbdirect.co.uk 195.173.25.
ipop3d:         192.168.0.
in.ftpd:        192.168.0.
in.timed:       192.168.0.
```

- Can base permissions on full/partial domains or addresses

8.6 Introduction to Packet Filtering

- Allows you to protect your machine
  - ♢ As well as machines *behind* them
- Checks packet headers before acting on them
  - ♢ Can ignore, reject or accept packets
  - ♢ Makes decision based on source, destination, or packet type
    - ∗ Or a combination
- Set up using `ipchains` under kernel 2.2
  - ♢ Older kernels used `ipfwadm`
  - ♢ 2.4 kernel now uses `iptables` - possibly too early to trust

8.7 Basic Packet Filtering

- Two main considerations
  - ♢ Port Filtering
  - ♢ Host Filtering
- Block services you don’t need
- Limit services you *do* need to specific machines/networks

8.8 `ipchains`

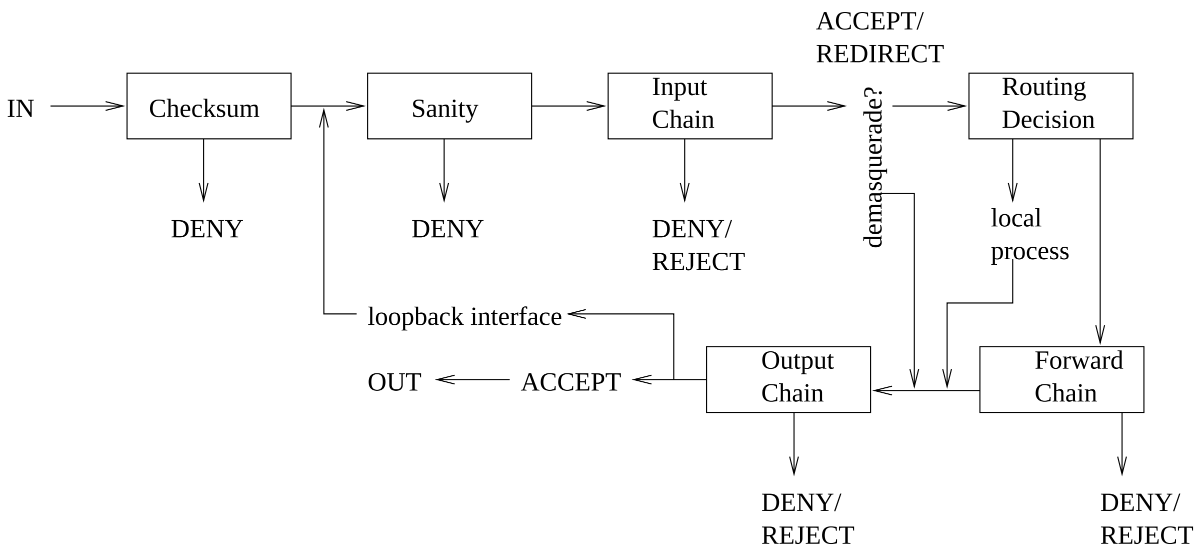
- Packet filtering set up using `ipchains`
- All the filtering is done in the kernel
  - ♢ Not by `ipchains`
  - ♢ `ipchains` just sets up/modifies the rules
- All packets entering and leaving are examined [1]

[1]Including loopback traffic which conceptually leaves the machine

8.9 `ipchains` Details

- Every packet goes through one or more ‘*chains*’
  - ♢ A ‘chain’ is a set of rules
  - ♢ Rules can accept, reject, or deny a packet
    - ∗ Can also send it to another chain
- Three default chains, *input*, *output*, *forward*
  - ♢ If a packet passes through a default chain without matching:
    - ∗ Fate is determined by the chain’s selected *policy*
    - ∗ Can be *Accept*, *deny*, or *reject*
  - ♢ If it reaches the end of a user defined chain
    - ∗ Carries on where it left off

8.10 `ipchains` schematic



- If installed, see `/usr/doc/ipchains-1.3.9/HOWTO.html` for much more detail
- *forward* is for packets routed to other hosts
  - ◇ Not covered here

8.11 `ipchains` Options

- Dealing with chains :

`-N`	Create a new chain
`-X`	Delete an empty chain
`-P`	Change the policy for a chain
`-L`	List the rules in a chain
`-F`	Flush (delete) all rules from a chain

- Dealing with rules :

`-A`	Append a rule to a chain
`-D`	Delete a single rule from a chain
`-I`	Insert a rule at some point in a chain

8.12 Options For Rules

- Use the following to specify packets to match

`-s`	Source address
`-d`	Destination address
`-p`	Protocol (`TCP`, `UDP`, `ICMP`)
`-j` *chain*	Jump to chain/action
`--sport`	Source Port
`--dport`	Destination Port

8.13 `ipchains` - Examples

- In most cases default chains will be sufficient
- To block all `ping` requests to our machine:

```
$ ipchains -A input -p icmp -s 0.0.0.0/0 --icmp-type echo-request -j DENY
$ ipchains -L input
Chain input (policy ACCEPT):
target  prot opt     source       destination    ports
DENY    icmp ------  anywhere     anywhere       echo-request
```

- To block outgoing `ping` packets:

```
$ ipchains -A output -p icmp -d 0.0.0.0/0 --icmp-type echo-request -j DENY
$ ipchains -L output
Chain output (policy ACCEPT):
target  prot opt     source       destination    ports
DENY    icmp ------  anywhere     anywhere       echo-request

$ ping -c1 landlord
PING landlord.gbdirect.co.uk (192.168.0.129): 56 data bytes
ping: sendto: Operation not permitted
ping: wrote landlord.gbdirect.co.uk 64 chars, ret=-1

--- landlord.gbdirect.co.uk ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

- Very simple examples but they show the principle

8.14 Removing Rules

- Rules can be removed by *number*, e.g. to delete the first rule in the *input* chain:

```
$ ipchains -D input 1
```

- or definition, e.g. delete the *first* matching rule:

```
$ ipchains -D output -p icmp -d 0.0.0.0/0 --icmp-type echo-request -j DENY
```

- To clear an entire chain use:

```
$ ipchains -F chainname
```

- If no `chainname` is given, it clears all chains

8.15 Implementing ipchains

- The rules are normally set up in the machine's 'init scripts'
- Typically by creating a script in `init.d` that is run just before networking starts
  - ◊ Example in section 16.19
- Ensure you flush existing rules first (just in case):

```
$ ipchains -F
```

- Generally start with the DENY rules then add what you want
- Maximum security

8.16 Save and restore

- Often useful to create a firewalling *'config file'*
- `ipchains-save` outputs a text file you can store

```
[Setup firewall rules as you want]
$ ipchains-save > /etc/ip.rules
Saving 'input'.
Saving 'forward'.
Saving 'output'.
$
```

- Can reinitialise your firewalling with `ipchains-restore` and your 'config file', e.g.

```
$ ipchains-restore < /etc/ip.rules
$
```

- Usually done in a startup script

8.17 `ipchains` setup script

- A sample script may look like:

```
#! /bin/sh
# Script to control packet filtering.
# If no rules, do nothing.
# Altered from the ipchains HOWTO
[ -f /etc/ipchains.rules ] || exit 0
case "$1" in
    start)
        echo -n "Turning on packet filtering:"
        /sbin/ipchains-restore < /etc/ipchains.rules || exit 1
        echo "."
        ;;
    stop)
        echo -n "Turning off packet filtering:"
        /sbin/ipchains -X
        /sbin/ipchains -F
        /sbin/ipchains -P input ACCEPT
        /sbin/ipchains -P output ACCEPT
        /sbin/ipchains -P forward ACCEPT
        echo "."
        ;;
  restart)
        $0 stop
        $0 start
        ;;
    *)
        echo "Usage: $0 {start|stop|restart}"
        exit 1
        ;;
esac
exit 0
```

8.18 Real World `ipchains`

- Connect out to a host but not in

```
$ ipchains -A input -d 192.168.0.131/32 -p TCP -y -j DENY
```

- `-y` limits matching to packets with the SYN bit set
  - ♢ Used when establishing connections
- No-one can open a connection from `192.168.0.131`
  - ♢ Can still connect to it from here . . .

8.19 Port Scanning with `nmap`

- `nmap` determines:
  - ♢ Which hosts are up on a network
  - ♢ What services each host offers
- Multiple scanning modes for different protocols
- Supports performance and reliability features:
- Flexible target and port specification
- You will probably have to download it or install it
- Examples:
  - ♢ `nmap 192.168.0.0/24`
  - ♢ `nmap -sS 192.168.0.1`
  - ♢ Refer to detailed documentation (later) for explanations

8.20 `nmap` Performance and Reliability Features

- Dynamic delay time calculations
- Packet timeout and re-transmission
- Parallel port scanning
- Detection of down hosts via parallel pings.

8.21 `nmap` Target and Port Specification

- Decoy scanning
- Determination of TCP sequence predictability characteristics
- Output to machine parseable or human readable log files.

8.22 Running `nmap`

- Should be run as root whenever possible
  - ♢ Not setUID
  - ♢ Some ordinary user functionality
- Lists key ports on scanned machine(s)
- Always gives the port's :
  - ♢ "Well known" service name
  - ♢ Number
  - ♢ State
  - ♢ Protocol

8.23 Reporting the State of Ports

State is either:

1. Open
   - Will accept() connections
2. Filtered
   - A firewall/filter, or other network obstacle is covering the port, preventing nmap from determining whether it is open
3. Unfiltered
   - Known to be closed, with no firewall/filter interference detected
   - The normal case

8.24 Reporting Other Details with `nmap`

Depending on options, `nmap` may also report:

- OS in use
- TCP sequencability
- Usernames running programs bound to ports
- DNS name
- Whether host is a smurf address [2]

[2] A smurf attack is a denial-of-service attack where the attacker sends ping requests to the broadcast address, having faked the source address of the victim. The victim receives a deluge of ping reply packets. This can be amplified across subnets if skill is used.

8.25 Scanning Modes: Vanilla TCP & SYN

- Vanilla TCP connect() scanning (-t)
  - ♢ Fast
  - ♢ No privileges needed
  - ♢ Easily detectable & filterable
- TCP SYN (half open) scanning (-s)
  - ♢ SYN|ACK confirms listening port, RST sent to end connection
  - ♢ Less detectable (few sites log it)
  - ♢ Needs root

8.26 Scanning Modes: IP Fragments & Reverse ident

- SYN/FIN scanning using IP fragments (-f)
  - ♢ Splits TCP header over several packets
  - ♢ Won't beat systems that queue IP fragment (e.g. Linux's CONFIG_IP_ALWAYS_DEFRAG option), but lots can't afford its performance hit
  - ♢ N.B. Has been known to segmentation fault sniffers!
- Reverse-ident scanning (-i)
  - ♢ Exploits ident protocol (reveals username of process connected by TCP), e.g. to find servers running as root
  - ♢ Needs full TCP connection to target port (-t)

8.27 Scanning Modes: TCP ftp proxy

- TCP ftp proxy (bounce attack) scanning
  - ♢ Uses an ftp proxy to send files to a 3rd server
  - ♢ Can be used to post virtually untraceable mail/news, provoke buffer overlows, fill up disks, etc
  - ♢ Proxy can scan ports from inside its firewall then send arbitrary data to open ones
  - ♢ Hard to trace
  - ♢ Can bypass firewalls
  - ♢ Slow
  - ♢ Useless against FTP servers which disable the proxy "feature"

8.28 Scanning Modes: UDP raw ICMP port unreachable

- More difficult than TCP scanning, because ports don’t send acknowledgements to probes
- Most closed ports do, however, send ICMP_PORT_UNREACH
- Revealing which are open by exclusion
- Needs re-transmission of lost packets
- Because neither UDP packets nor ICMP errors are guaranteed to arrive
- Need confirmed closure to avoid false positives
- Slow because RFC1812 suggests limits to ICMP error message rates

8.29 Some other `nmap` Scanning Modes

- ICMP scanning (ping-sweep)
- TCP FIN, Xmas, or NULL (stealth) scanning
- TCP ACK and Window scanning
- TCP Ping scanning
- Direct (non portmapper) RPC scanning
- Remote OS Identification by TCP/IP Fingerprinting, and

8.30 `nmap` Documentation

- The manual page `man nmap`
  - ◇ Most complete and up-to-date source
  - ◇ An online HTML version:
    `http://www.insecure.org/nmap/nmap_manpage.html`
- The home page
  `http://www.insecure.org/nmap/`
- Background info on how nmap uses TCP/IP fingerprinting for remote OS detection:
  `http://www.insecure.org/nmap/nmap-fingerprinting-article.html`

8.31 Basic `nmap` Howto

```
From lamontg@raven.genome.washington.edu Sun Apr 11 03:03:15 1999
Date: Mon, 5 Apr 1999 16:50:23 -0700
From: Lamont Granquist <lamontg@raven.genome.washington.edu>
To: nmap-hackers@insecure.org
Subject: NMAP guide

-------------------------------------------------------------
NMAP does three things.  First, it will ping a number of hosts to
determine if they are alive or not.  Second, it will portscan hosts to
determine what services are listening.  Third, it will attempt to
determine the OS of hosts.

Of course NMAP is very configurable, and any of these steps may be
omitted, (although portscanning is necessary in order to do an OS scan),
and there are multiple ways to accomplish most of these, and many command
line switches to tweak the way that NMAP operates.

Target Selection

You can specify NMAP targets both on the command line or give a list of
targets in a filename with the -i option.  As the NMAP help documentation
suggests you can use the hostname/mask method of specifying a range of
hosts (cert.org/24 or 192.88.209.5/24) or you can give a explicit IP range
(192.88.209.0-255).  The '24' in 'cert.org/24' is the number of bits in
the mask, so /32 means "just that host", /24 means "the 256 addresses in
that Class C", /16 means "the 65536 addresses in that Class B", /8 would
be "the 2^24 addresses in that Class A" and /0 would scan all possible
(IPv4) 2^32 IP addresses.

Ping Scans

The default behavior of NMAP is to do both an ICMP ping sweep (the usual
kind of ping) and a TCP port 80 ACK ping sweep.  If an admin is logging
these this will be fairly characteristic of NMAP.  This behavior can be
changed in several ways.  The easiest way is, of course, to simply turn
off ping sweeps with -P0.

If you want to do a standard ICMP ping sweep use -PI.  If you are trying
to get through a firewall, though, ICMP pings will likely be blocked and
using packet filtering ICMP pings can even be dropped at the host.  To get
around this NMAP tries to do a TCP "ping" to see if a host is up.  By
default it sends an ACK to port 80 and expects to see a RST from that port
if the host is up.  To do only this scan and not the ICMP ping scan use
-PT.  To specify a different port than port 80 to scan for specify it
immediately afterwards, e.g. -PT32523 will ACK ping port 32523.  Picking a
random high-numbered port in this way may work *much* better than the
default NMAP behavior of ACK pinging port 80.  This is because many packet
filter rules are setup to let through all packets to high numbered ports
with the ACK bit set, but sites may filter port 80 on every machine other
than their publically accessable webservers.  You can also do both an ICMP
ping scan and an ACK scan to a high numbered port with, e.g. -PB32523.
However, if a site has a really, really intelligent firewall that
recognizes that your ACK packet isn't part of an ongoing TCP connection it
might be smart enough to block it.  For that reason, you may get better
results with a TCP SYN sweep with -PS.  In this case, scanning a
high-numbered port will probably not work, and instead you need to pick a
```

port which is likely to get through a firewall. Port 80 is not a bad pick, but something like ssh (port 22) may be better.

So the first question to ask yourself is if you care about wasting time scanning machines which are not up and if you care about getting really complete coverage of the network? If you don't care about wasting time and really want to hit all the machines on a network, then use -P0. Pinging machines will only cause you to have more of a signature in any log files and will eliminate machines which might possibly be up. Of course, you will waste time scanning all the IP numbers which aren't assigned.

If you do ping machines, an ICMP ping sweep is probably more likely to be missed or ignored by system administrators. It doesn't look all that hostile. If you think you're up against a firewall you should experiment with which kinds of pings seem to get through it. Do ICMP pings work at all? Can you ping thier webserver? If not, then don't bother with ICMP pings. Can you ACK ping thier webserver? If not, then you have to go with SYN pings.

What if all you want to do is a ping scan? Then use -sP.

Port Scanning

The vanilla scan is a TCP connect() scan (-sT). These are loggable. You probably don't want to do these.

SYN scans (-sS) are the workhorse of scanning methods. They are also called "half-open" scans because you simply send a SYN packet, look for the return SYN|ACK (open) or RST (closed) packet and then you tear down the connction before sending the ACK that would normally finish the TCP 3-way handshake. These scans don't depend on the characteristics of the target TCP stack and will work anytime a connect() scan would have worked. They are also harder to detect -- TCP-wrappers or anything outside of the kernel shouldn't be able to pick up these scans -- packet filters like ipfwadm or a firewall can though. If a box is being filtered NMAP's SYN scan will detect this and report ports which are being filtered.

FIN (-sF), NULL (-sN) and XMAS (-sX) scans are all similar. They all rely on RFC-compliance and as such don't work against boxes like Win95/98/NT or IRIX. They also work by getting either a RST back (closed port) or a dropped packet (open port). Of course, the other situation where you might get back a dropped packet is if you've got a packet filter blocking access to that port. In that case you will get back a ton of false open ports. A few years back these kinds of scans might have been stealthy and undetectable. These days they probably aren't.

You can combine any of the SYN, FIN, NULL or XMAS scans with the (-f) flag to get a small fragment scan. This splits the packet which is sent into two tiny frags which can sometimes get through firewalls and avoid detection. Unfortunately, if you're not running a recent version of an open source O/S (Linux or Net/Open/FreeBSD) then you probably can't frag scan due to the implimenation of SOCK_RAW on most unixes (Solaris, SunOS, IRIX, etc). See Fyodor's NMAP portability chart to see if -f is supported on your platform.

For the initiated out there, you could modify libpcap to allow you to send packets in addition to sniffing them by opening the packet capture device rw instead of ro. Then you need to build a link-layer (probably ethernet) header and then you could impliment your own frag scanner. For bonus points impliment all of the different SYN, FIN, NULL and XMAS scans *and* allow for sending the fragments out in reverse order (which helps for getting through firewalls). This hasn't been done (yet) in NMAP due to the fact that NMAP needs to support multiple different link layer interfaces (not just ethernet) and needs code for dealing with ARP. If anyone wants to code this up, I'm sure that people would appreciate it.

UDP scanning (-sU) in NMAP has the same problem as FIN scans in that packet filtered ports will turn up as being open ports. It also runs extremely slowly against machines with UDP packet filters.

Another type of scan is the bounce scan (-b <ftp_relay_host>) which, if there is insufficient logging on the ftp host you're using to bounce, is completely untraceable. Recent FTP servers shouldn't let you do these kinds of scans.

The last scanning option that I'm going to mention is identd scanning (-I) which only works with TCP connect scans (-sT). This will let you know the owner of the daemon which is listening on the port. Provided, of course, that the site is running identd and is not doing something intelligent like using a cryptographic hash (i.e. pidentd -C). You *have* to make complete 3-way TCP handshakes for this to work, so this is not very stealthy. It does, however, give you a lot of information. It only works against machines that have port 113/auth open.

Source IP Deception

You can also take advantage of the fact that you can change your source address. The simplest way to do this is with -S <ip>. If you are on a broadcast ethernet segment you could change your source address to an IP which doesn't exist and then you simply sniff the network for the reply packets. And if you are not on a leaf node/network then as long as the reply packet will get routed by you, you can use it. To turn this on its head: the next time you get scanned, do a traceroute on the machine that scanned you. Any of the machines on any of the networks that those packets went through could have been the machine which was *really* scanning you.

The other deceptive measure is to use decoy scans. You spoof a ton of scans originating from decoy machines and insert your IP in the middle of it somewhere. The admin at the site you are scanning is presented with X number of scans and no way to determine which one actually did it. For bonus points, combine this with the previous tactic and spoof an IP address which doesn't exist. If you don't spoof your own IP address make sure to use "likely" decoys -- use machines which were connected to the net at the time you made your scans and don't use sites like www.microsoft.com. Ideally you want a lot of linux boxes as decoys. The more decoys the better, but obviously the slower the scan will go.

[QUESTION: do decoy/spoof scans also decoy/spoof the ping scan? can you combine decoy scans and "ME" spoofing like this? does a decoy/spoof scan also decoy/spoof the OSscan?]

OS scanning

This is the -O option. To use it requires one open and one closed port. The closed port is picked at random from a high-numbered port. Machines which do packet filtering on high-numbered ports will cause problems with

```
OS detection (many sites will filter packets to high numbered ports which
don't have the ACK bit set).  Also excessive packet loss will cause
problems with OS detection.  If you run into trouble try selecting an open
port which isn't being served by inetd (e.g. ssh/22 or
portmap/rpcbind/111).

OS scanning also reports the TCP sequence number prediction vulnerability
of the system.  If you're 31337 you will be able to use this to exploit
trust relationships between this machine and other machines.  There's a
reasonably decent phrack article on this in phrack P48-14, but you should
beware that it isn't this easy -- you need to worry about ARP (what's
that?  how does it work?  i suggest familiarizing yourself with tcpdump)
and if you're trying to exploit rsh/rlogin you need to worry about
spoofing the authorization connection as well.


--
Lamont Granquist                       lamontg@genome.washington.edu
Dept. of Molecular Biotechnology       (206)616-5735  fax: (206)685-7344
Box 352145 / University of Washington / Seattle, WA 98195
PGP pubkey: finger lamontg@raven.genome.washington.edu | pgp -fka
```

8.32 `ssh/scp/sshd`

- For fuller description see "System Daemons" module
- `ssh` is a secure replacement for `rsh`, `telnet` etc.
  - ♢ Older programs send passwords in plain text; disastrously easy to sniff
- Can also encrypt traffic over arbitary ports
  - ♢ Enables secure POP for example
- Very secure if set up correctly
- `sshd` is the server part
- Answers requests from `ssh` clients

8.33 `ssh` principles

- Has various authentication methods
  - ♢ Some not much better than `rsh`!
- However it does ensure all traffic is encrypted
  - ♢ Stops people sniffing your password
- When you connect the server sends two things
  - ♢ The host's public key
  - ♢ The server's public key
- Used as the basis for authentication challenges and encryption of the session
- Lots of documentation: `http://www.ssh.com`, etc

8.34 Use of `ssh`

- Should be used in preference to `telnet` etc.
- Will even support remote use of X clients (inserts a proxy)
- Failure to use it is a severe mistake
- Clients are available for various host OS types
- Once set up is very easy and simple
- Examples:
  - ◊ `ssh -l mikeb landlord`
  - ◊ `scp mikeb@tiger:Projects/sample1.txt davef@landlord:`

8.35 Exercises

1. `ipchains`
   (a) Use ipchains to set up the following configurations. In each case you should first set up the system by hand, check it. Then set it up so that the firewall rules are in place when the machine reboots.
      i. Block all incoming ICMP packets
      ii. Block only incoming ICMP ‘echo-request’ packets
      iii. Block all incoming telnet connections
      iv. Block *all* telnet connections
      v. Block all outgoing web requests (Port 80)

2. `nmap`
   (a) Use nmap to scan another host in your training room using the following scanning modes one after another
      i. Vanilla TCP
      ii. TCP SYN (half open) scanning,
      iii. TCP FIN, Xmas, or NULL (stealth) scanning,
      iv. TCP ftp proxy (bounce attack) scanning
      v. SYN/FIN scanning using IP fragments (bypasses some packet filters),
      vi. TCP ACK and Window scanning,
      vii. UDP raw ICMP port unreachable scanning,
      viii. ICMP scanning (ping-sweep)
      ix. TCP Ping scanning
      x. Direct (non portmapper) RPC scanning
      xi. Remote OS Identification by TCP/IP Fingerprinting
      xii. Reverse-ident scanning.

      N.B. You may have to use the nmap man page to look up appropriate options for some of these scanning modes.

   (b) Arrange so that each member of your group tries to decoy scan every other machine in the group.

      Try to detect which of your colleagues lies behind each of the decoy addresses used against your machine.

8.36 Solutions

1. (a) The following are the list of rules needed to satisfy each situation. You should flush the chains before each one (`ipchains -F`).

    i. `ipchains -A input -p icmp -j DENY`

    ii. `ipchains -A input -p icmp --icmp-type echo-request -j DENY`

    iii. `ipchains -A input -p tcp -d 127.0.0.1 --dport telnet -j DENY`
    `ipchains -A input -p tcp -d 192.168.0.131 --dport telnet -j DENY`

    iv. `ipchains -A output -p tcp -d 0/0 --dport telnet -j DENY` `ipchains -A input -p tcp -s 0/0 --dport telnet -j DENY`

    v. `ipchains -A output -p tcp -d 0/0 --dport www -j DENY`

Module 9

Job Control Tools

Objectives

At the end of this section, you should be able to understand and use these facilities:

- Sending signals (inc termination) to processes
- Change the priority given to running jobs
- Run processes after detachment or logout
- Run programs at specified times
- Run programs when the system is lightly loaded
- Run programs at *regularly* specified times

9.1 Introduction

- The tools discussed in this section have the following uses:

`kill`	send signals to a process
`nice`	change scheduling priority
`nohup`	run a job independently
`at`	run a task at a specified time
`batch`	operate a job queue
`cron`	run a task on a regular basis

9.2 Background Jobs

- Normal Shell Operation

```
$ command arg1 arg2 . . .
< stdout and stderr of command >
< Some time later >
$
```

- Background Job

```
$ command arg1 arg2 . . . &
23475
$ < now able to do something else >
< possible corruption of screen >
```

- Better Still

```
$ command arg1 arg2 . . . 1> log
2>errlog &
17456
$
```

9.3 `kill`

- Sends signals to processes, e.g. termination
- Syntax:

```
kill [signal] process_id
```

- Built-in to `bash`
- signal may be either signal number or name. e.g.

```
kill -1 3145
kill -HUP 3145
```

- `kill -l` gives a list of signal names/numbers
  - ◊ N.B. this is a lower case L, `kill -1` would send the `TERM` signal to all processes you own!
- Process ID can be specified as either:
  - ◊ PID (listed by `ps`)
  - ◊ Job number, e.g.

    ```
    $ kill -9 %2
    ```
- `kill -9 PID` can be used to forcibly kill a process

9.4 The `nice` Command

- `nice` Changes the priority of a command
- Syntax:

```
nice [options] command-line
```

- A user may *lower* a command's priority, by increasing its `nice` number
- Here is an example

```
nice -20 bigjob 1>nice.out 2>&1 &
```

- Note that this increases the `nice` *number*
- Only the super-user may increase the priority of a job, by *decreasing* the `nice` number:

```
nice --20 important_job &
```

- Legit `nice` numbers range from -20 to 20

9.5 Hang Up (`nohup`)

- `nohup` keeps a command running after detatchment or logout
- Usually, a detached job should be allowed to finish, even if the top level shell terminates
- `nohup` is the mechanism to achieve this
- Typing:

  ```
  nohup bigjob &
  ```

  means that:
- stdout and stderr are both appended to the file `nohup.out`
- Thus, a job can continue, even after logout

9.6 Execute programs at specified times (`at`)

- `at` executes a shell script at a time you specify
- Syntax:

  `$ at [`*options*`]` *time* `[`*date*`]` +*increment*
- `stdin` is scanned for commands to execute, e.g.

```
at 2300
at> fetchmail
<CTRL>+D
```

- Some more a examples of how to specify time and date:

```
at 11pm
at 1am
at 5am tomorrow
at 08:00 Sep 12
at now
at now + 4 hours
at now + 1 day
at now + 2 months
at now + 3 years
at 5:30pm Thursday
```

- Commands are executed in the current environment at the given time
- `stdout` and `stderr` are sent as mail

9.7 Options and commands related to `at`

- `at` belongs to a family of utilities for managing time-specified commands

Command	Purpose
`atq`	Display list of queued `at` commands
`atrm`	Remove queued `at` commands
`batch`	Schedule jobs at low CPU loading

- All of these individual commands can be run as `at` options:

Option	Purpose
`-l`	Display list of queued `at` commands
`-d`	Remove queued `at` commands
`-b`	Schedule jobs at low CPU loading
`-f`*`file`*	Specify script file in command-line
`-m`	Send mail after running `at`, whatever the `stdout` or `stderr`

- UNIX NOTE:
  On System V, two files control the use of `at`
  - ◇ `/usr/lib/cron/at.allow`
  - ◇ `/usr/lib/cron/at.deny`

9.8 Running commands regularly (`crontab`)

- `crontab` lets you submit job lists at regular times using the `crond` daemon
- 2 syntax formats:

```
$ crontab filename
$ crontab [-u username] [options]
```

- Options, etc:

Command	Purpose
`crontab` *myfile*	Install contents of *myfile* (`stdin` if no file specified) in appropriate directory
`crontab -r`	Remove the crontab for the current user
`crontab -l`	List (on `stdout`) current user's `crontab`. (might be useful for editing a cron table)
`crontab -d`	Delete your crontab file
`crontab -e`	Run a text editor on your crontab file

Table 9.1: `crontab` usage

- Examples of Crontab Entries

```
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
# LW poll client1  for mail
0 10,12,16,18 * * 1-5 root /www/bin/client1_poll
LW 16/09/1998 - Hylafax cron stuff ...
0 0 * * *  root /usr/local/sbin/faxqclean
10 0 * * * fax /usr/local/sbin/faxcron -info 7
```

9.9 Summary

In this section we have examined:

- Detached jobs
- Altering Process Priorities (`nice`)
- Postponing Jobs (`at`)
- Batch Processing
- Regularly Scheduled Jobs (`crontab`)

9.10 Job Control Exercises

1. Create the following shell-script called `alltrue` in your home directory:

   ```
   #!/bin/sh
   while [ 1 ]
   do
           foo = "foo"
   done
   ```

   As you can probably tell this doesn't do anything useful apart from continually doing nothing!

   Run this process in the background and check its nice level by using `ps -l`, you should see something like the following:

   ```
   FLAGS  UID PID  PPID PRI NI SIZE RSS WCHAN STA TTY TIME COMMAND
   100000 504 4033 3064 16  0  1148 608 0     R   a1  0:01 sh ./alltrue
   ```

   The process is running at it's default nice level of 0, try running the process with a lower scheduling priority, i.e. a higher nice value. Check that the nice level has changed using `ps -l` (As above.)

2. What happens if you try to higher the priority, ie by lowering the nice value. Why?

3. Start a shell and start the process alltrue in the background like above. Find out its process ID (The `PID` column in a `ps -l`). You can kill this process by doing a `kill -HUP PID`. Now start the process using `nohup`, and try and kill it. Also note its nice level.

4. Schedule a job using `at` to run in 5 minutes time that will echo some text to a file. When 5 minutes has passed used `ls -l` to check that the file was created.

5. Repeat exercise 4 but using `cron` and `crontab`.

9.11 Job Control Solutions

1. —
2. Only the root user is allowed to raise the priority of a process. Trying to give a process a negative nice value (high priority) will give you a `"permission denied"` error message.

3.
   ```
   at now + 5 minutes
   at> echo foo > new_at_file
   ```
4. `crontab myfile` where *myfile* contains:

   ```
   hh mm dd MM * echo foo > new_cron_file
   ```

   where *hh mm* is the hour and minutes that you want the job to run at, *dd* is the day of the month and *MM* is the month.

Module 10

Overview

Objectives

Having completed this module, you will have an overview of a Linux system, including its:

- Underlying philosophy
- System layering - kernel vs. applications
- Core services
- Multiuser and timesharing facilities
- File System
- Network Services
- Desktop and X windowing system

10.1 Generic Features of Unix

- Component-based systems
- Very popular with technically skilled
- Not ‘solution’ oriented
- Building blocks not the building
- Highly network-aware
- Robust, powerful, reliable

10.2 Linux — The Kernel of a System

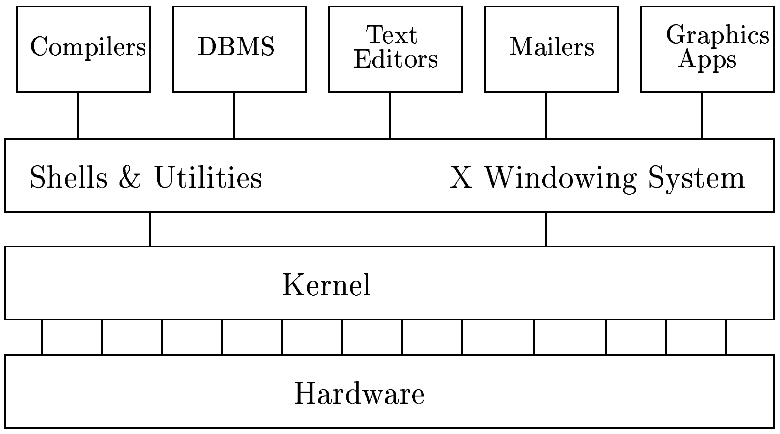


Figure 10.1: kernel-layering

- What is *called* Linux is actually a collection of components from many sources
  - ♢ freely copiable, under ‘open source’ licences
- Linux is, strictly, just the *kernel* which provides:
  - ♢ A common interface between *user process* and hardware
  - ♢ Minimal functions to user applications, i.e. system calls
  - ♢ Scheduling

10.3 Fundamental Characteristics of Linux

- Multi-tasking
- Multi-user access
- Multi-processor
- Architecture independence
- POSIX 1003.1 plus basic System V and BSD
- Protected memory mode
- Multiple filesystem types
- Comprehensive networking (TCP/IP and others)
- Multiple executable formats (MS-DOS, iBCS UNIX, SCO, etc)

10.4 Multiuser Multitasking and Time-sharing

- Designed as a multi-user system
  - ♢ Each user's shells, apps and commands are separate processes
  - ♢ Number of simultaneous users limited only by:
    - ∗ CPU speed and available memory
    - ∗ Min. response times required by users/apps
- Multi-tasking:
  - ♢ Many jobs can be under way at the same time
  - ♢ Jobs truly *simultaneous* on multi-cpu
- Time-sharing:

  A single cpu is shared by all processes
  - ♢ Processes exec briefly, passing cpu to others
  - ♢ *Process switches* occur in miliseconds or less
  - ♢ Kernel gives process a sense of total control

10.5 Protected memory mode

- Uses the processor’s protection mechanisms
- Prevent access to memory already allocated to kernel or other processes
- Bad programs can’t crash the system
  - ♢ Theoretically

10.6 Multiple Filesystem Types

- Native FS is ext2 (Second Extended File System)
  - ♢ File names up to 255 chars
  - ♢ More secure than conventional UNIX
- Others include:
  - ♢ MS-DOS (FAT16), VFAT, FAT32
  - ♢ ISO9660 (CD-ROM)
  - ♢ HPFS (OS/2)
  - ♢ NTFS (Windows NT)
  - ♢ UPS, SysV and other proprietory UNIX
  - ♢ NFS (Unix network file system)
  - ♢ SMB / CIFS (MS Windows file sharing)

10.7 The Many Faces of a GNU/Linux System

- The user may see up to five aspects of Linux:
  - ♢ the *filesystem*
  - ♢ *processes*
  - ♢ the *shell*
  - ♢ the X *windowing system*
  - ♢ *Inter-Process Communication* (IPC)
- The system is very highly configurable
- Different users may experience totally different views of the same system
- Multiple simultaneous users are normal
  - ♢ Linux is designed from the ground up as a *multi-user system*, NOT a ‘personal’ system

10.8 The Filesystem

- The filesystem contains all data in the system
- A name in the filesystem can refer to:
  - ♢ a *data file*, which can be:
    - ∗ a *plain file*
    - ∗ a *directory*
  - ♢ a *device* (disk, tape etc.)
  - ♢ internal memory
  - ♢ OS information (the *proc* system)
- Directories are groups of files
  - ♢ Grouped in hierarchical *trees*
- Files are fully specified with their *pathname*
- An original Unix structure; copied by most OSs

10.9 Filenames

- Maximum length depends on filesystem type
  - ♢ Most allow up to 255 characters
- Can use almost any character in a filename, but avoid ambiguity by sticking to:
  - ♢ (A-Z) Uppercase letters
  - ♢ (a-z) Lowercase letters
  - ♢ (0-9) Numbers
  - ♢ (.) Full-stop
  - ♢ (,) Comma
  - ♢ (_) Underscore
  - ♢ (-) Hyphen
- Should convey meaningful info about contents
- Type longer filenames using completion for:
  - ♢ Filenames
  - ♢ Pathnames
  - ♢ Commands

10.10 Filename Extensions and File Types

- Filenames *don't* determine other attributes of file,

  i.e. do not, *automatically*, cause command interpreters to treat them in a particular way

- However:
  - ♢ Extensions can enable meaningful naming and automatic file manipulation
  - ♢ C compilers and some other programs *do* depend on specific file extensions to carry out particular tasks

- Common conventions for extensions:

Filename	Meaning of Extension
program.c	C programming source file
program.o	Object code
program.sh	Shell executable
letter.txt	Text file of a letter
letter.ps	Postscript version of same letter file
letter.ps.gz	`gzip` compressed version of same
letter.tgz	`tar` archive of same compressed by `gzip`
letter.tar.gz	Another, more common, way of naming `*.tgz`
letter.Z	Same file compressed with outdated `compress` utility

Table 10.1: Common conventions for filename extensions

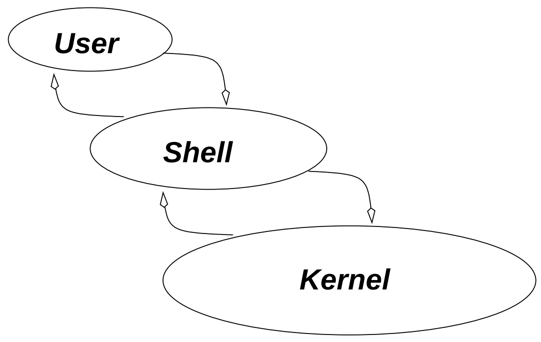
10.11 Hidden Filenames

- Filenames beginning with a full-stop are *hidden*
- Typically used:
  - ♢ To hide personal configuration files
  - ♢ To avoid cluttering dirs with rarely used files
- Every dir contains 2 special hidden files:

.	The current directory file
..	The parent directory file

10.12 The Shell (`bash`)

- A *shell* is a program that you interact with



- Can be any program, but is normally a *command interpreter*
- A command interpreter is usually started when you log in (but this is just one way)
- The 'standard' Linux command interpreter is a `Bourne` *shell* look-alike called `bash` [1]
- The command line syntax provided by `bash` enables manipulation of files & processes
- The command-line frightens beginners but is the preferred home of the skilled

[1]`Bash` has more functions than true `Bourne` shells; incorporating most of the innovations added by the `C` and `Korn` shells. `Bash` functions and flags differ between implementations of UNIX and Linux. The version of `bash` in current Linux releases tends to be the most fully functional `Bourne` shell around.

10.13 Key Features of the Bash Shell

- Command history
- Command aliasing
- Shell scripting
- Filename completion
- Command completion
- Command line editing (`emacs` and `vi` styles)
- Job control
- Key Bindings
- Directory stacking
- Tilde directory notation
- Help function, e.g.

```
$ help history
history: history [n] [ [-awrn] [filename]]
    Display the history list with line numbers.  Lines listed with
    with a ‘*’ have been modified.  Argument of N says to list only
    the last N lines.  Argument ‘-w’ means to write out the current
    history file;
```

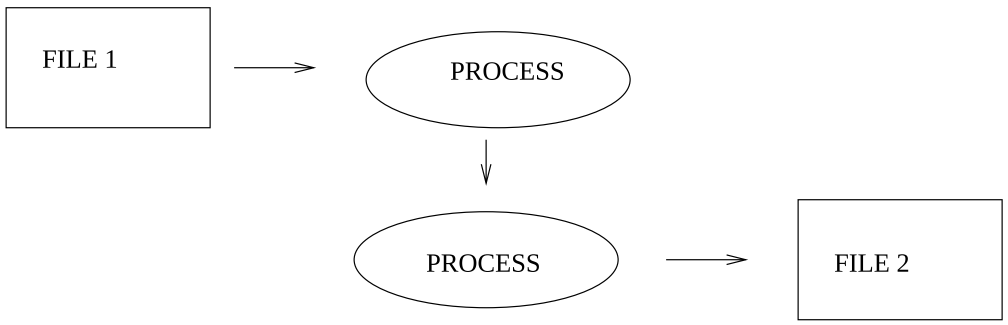
10.14 Interacting with a Linux 'Terminal'

- Linux can support any number of 'terminal' types
  - ♢ nowadays, monitor/keyboard combinations
  - ♢ previously, dumb terminals
  - ♢ occasionally, printers (debugging servers)
- Most will use the *console* or a windowed terminal, but if not:
  - ♢ Linux usually keeps a database of terminal capabilities in `/etc/termcap` [2]
  - ♢ If your terminal type is not recorded in `/etc/termcap`, you'll have problems running certain programs e.g.
    - ∗ cursor driven apps (`top`, `linuxconf`, `vi` etc)
  - ♢ The *environmental variable* `TERM` tells programs what terminal type you are using

[2] AT&T flavours of UNIX use `/usr/lib/terminfo` to store the same information and Linux can, if necessary.

10.15 Software Tools: The UNIX Philosophy

- True UNIX-like systems treat programs as *tools*
  - ♢ Each tool should:
    - ∗ Do just one thing well
    - ∗ Be generic (untied to specific applications)
  - ♢ For new jobs, build new tools
  - ♢ (Re-)combine, don't complicate old tools
- Linux can do this because it has:
  - ♢ two simple *objects*:
    - ∗ the file
    - ∗ the process
  - ♢ simple methods of *connecting*:
    - ∗ processes to files
    - ∗ processes to processes



10.16 Tasks/Processes

- A *program* is an execut*able* object, stored in a file
- A *process* is an execut*ing* object, i.e. [3]
  - ♢ an *instance* of a program currently being run
- Existing processes can '*fork*' to create other processes
  - ♢ the only way to make new processes
- A user may run multiple copies of same program
- Multiple users may run single/multiple copies
- System tracks *ownership* and *permission*

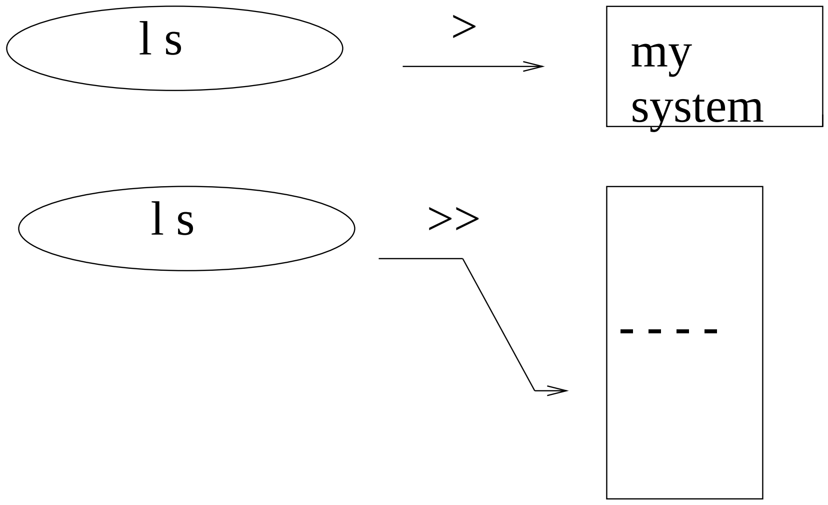
[3]Processes are often called *tasks*, as in 'multi-tasking'

10.17 Process Communication

- Processes may need to co-operate by
  - ♢ sharing files
  - ♢ signalling events
  - ♢ direct transfer of data
  - ♢ pipelines (data streams)
  - ♢ synchronising with each other
- Linux provides facilities for:
  - ♢ signals
  - ♢ shared memory
  - ♢ pipes, both named and unnamed
  - ♢ semaphores
  - ♢ and others
- Processes may use network connections for communication, permitting *client-server* model
  - ♢ Common for shared services like printing

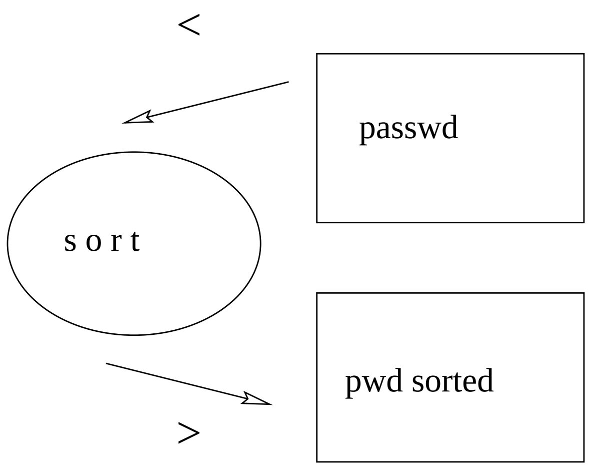
10.18 Re-directing I/O to and from Files

- Most processes will take input from the keyboard and output to the screen
- Both input and output streams can be *re-directed* to/from files
- Output to a file (creating or overwriting):
  `$ ls > my-system.txt`
- Appending output to a file: `$ who >> my-system.txt`



10.19 Re-directing I/O to and from Files (continued)

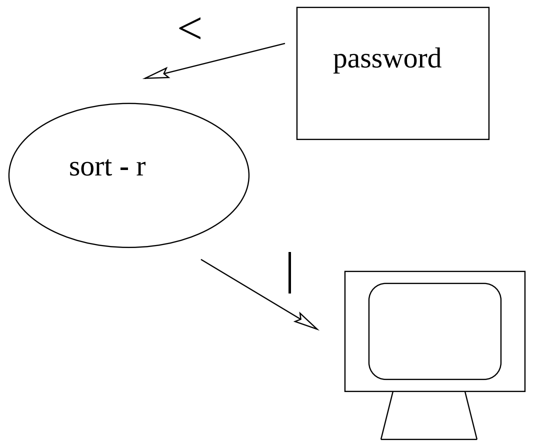
- Take input from one file, output to another: `$ sort < /etc/passwd > pwd.sorted`



10.20 Pipes & Tools

- Linux tools act as filters:
  - ♢ taking data from input streams, modifying it, sending it elsewhere
  - ♢ expecting data to come from other tools
  - ♢ producing output which *any* other tool can process, e.g. ASCII text
- One tool's output is connected to another's input:
  - ♢ *Indirectly*, via a file created by the first tool
  - ♢ *Directly*, via a *pipe* or *pipeline*
- For example, to page through a reverse-sorted version of your password file on screen:

```
$ sort < /etc/passwd | less
```



10.21 Linux as a Programming Environment

- *Hierarchical Filestore*
- Extensive set of *powerful tools*
  - ♢ for software production, admin and support
- A *common system interface*
  - ♢ only one set of procedures to learn
- Processes interface with *anonymous files*
  - ♢ programs output to files or devices identically
- *Modular architecture* provides for a completely customised OS, e.g.
  - ♢ An OS dedicated solely to graphics rendering
  - ♢ A general-purpose system on one floppy
- *Flexible user interface* allows for uniquely customised programming environments

10.22 Networking

- Linux is a network operating system.
- The Internet network protocols (TCP/IP) are implemented in the kernel
- Although other media are supported (e.g. radio, infra-red), links are usually across:
  - ♢ Ethernet
  - ♢ Serial Line (Point-to-point)
- Proprietory file/print serving protocols supported:
  - ♢ Appletalk
  - ♢ DECNET
  - ♢ IPX / Novell Netware
  - ♢ SMB / CIFS (MS Windows/NT)

10.23 TCP/IP

- A suite of Internet-standard protocols and apps for managing data transfers
- Depicted as a ‘stack’
  - ♢ hardware and transport control protocols at the bottom
  - ♢ user applications (e.g. browsers) at the top
- Client-server apps provide facilities for:
  - ♢ Remote login
  - ♢ File transfer
  - ♢ Resource sharing (e.g. expensive peripherals)
  - ♢ Remote command execution
  - ♢ Email (internet/intranet/extranet)
  - ♢ Web browsing

10.24 Documentation

- Copious, but fragmented and/or duplicated

Programmer's Manual `/usr/man`	The classic '*man pages*', first stop for skilled users, worth learning
`info` pages	hypertext browsable texts, often identical or updated versions of man pages
`/usr/doc/`*`program-name`*	ascii/html docs installed with the named program
Howtos	Tutorials on Linux-related topics, available on-line if installed (usually in `/usr/doc`)
www	Recently-released programs are usually documented on authorised web sites, many (including older tools) are documented by third-party sites

Table 10.2: Sources of Linux Documentation

- Linux man pages divided into sections:

  1. User Commands
  2. *System calls*
  3. Subroutines (inc library routines)
  4. Devices (inc network interfaces)
  5. File Formats
  6. Games
  7. Miscellaneous
  8. System Administration

- The `apropos` command word searches the description line in man pages. Thus:

```
apropos printer
```

  will find man pages relating to printers, e.g.

```
lp (4)   - line printer devices
lpd (8)  - line printer spooler daemon
lprm (1) - remove jobs from the line printer spooling queue
```

10.25 Using the *man pages* (On-Line Manual)

- Use `man` to see man pages on a named command, e.g

```
$ man date
```

- The result should be something like:

```
DATE(1)
NAME date - print or set the system date and time
SYNOPSIS
date [-u] [-d datestr] [-s datestr] [--utc]
[--universal] [--date=datestr] [--set=datestr]
[--help] [--version] [+FORMAT] [MMDDhhmm[[CC]YY][.ss]]
```

- `DATE(1)` Shows page is in manual section 1
- To view a page from a certain section use:

  `$ man -S` *`section-number command-name`*

- Square brackets surround optional arguments

```
ls  [-abcdfgiklmnpqrstuxABCFGLNQRSUX1]
```

10.26 Overview Exercises

1. *Logging in*

   (a) Practice logging in by typing your username and password in response to the `Login:` and `Password:` prompts, e.g. [4]

   ```
   Login:  mikeb
   Password:
   $
   ```

   Once you have logged in, the `bash` prompt (`$`) is printed; indicating the shell is ready to take commands.

   (b) Log out, by typing `exit` at the `$` prompt. You should get a new `Login:` prompt, at which you can login again.

2. *Changing password*

   (a) Set yourself a new password using the `passwd` command. Run the command by typing `passwd`, followed by a `<RETURN>` .

3. *Navigating Man Pages*

   (a) Type `man man` to open the man page which details how to use the `man` command

   (b) Press the `h` (help) key, which opens a "Summary of Less Commands", including all the keystrokes you need to navigate a man page

   (c) Make sure you can quit this page (by typing `q`) and quit the man page (by typing `q` again). When you get back to the shell prompt, repeat the first 2 steps to open the "Summary of Less Commands" from the `man` man page.

   (d) Use the "Summary of Less Commands" to make sure you know how to do the following bits of navigation inside a man page:

      i. Move to the top and bottom of the man page
      ii. Move up and down one screen of text
      iii. Move up and down one line of text
      iv. Search forward for a *pattern* (e.g. a word)
      v. Search backwards for a *pattern*
      vi. Repeat a forward *pattern* search using one key
      vii. Repeat a backward *pattern* search using one key
      viii. Move to a specific line number

   (e) With a partner, test each other on how well you can navigate the `man` man page, e.g. set each other target locations or words to go to.

4. *Invoking the Right Man Pages*

   (a) Using the `man` man page, find the command string you need to use to get the following:

      i. A list of man pages whose description lines contain details about the 'whatis' database

[4] A `<RETURN>` is required after each input, to tell the shell that you have finished typing and it should start processing your request. Note that the password is not displayed on the screen; to keep it secret

   ii. A list of man pages containing the string ‘cdrom’ [5]
   iii. A list of man pages from a specific section (e.g. 1) of the manual, whose description lines contain ‘print’

(b) Practice using these flags to find and view man pages which deal with computer keywords your partner sets for you (and vice versa), e.g.

   i. bitmap formats like jpg, gif, xpm, bmp
   ii. communications concepts like modem, serial, telnet, pcmcia, ppp
   iii. filesystems like NFS, ext2, FAT, vfat, msdos, samba

5. *Finding Out About Your System and Users*

   (a) Type the following commands. Identify what each of them tells you about your system.

      i. `$ whoami`
      ii. `$ who am i`
      iii. `$ users`
      iv. `$ who`
      v. `$ w`
      vi. `$ date`
      vii. `$ cal 8 1999`
      viii. `$ cal 9 1752` [6]
      ix. `$ df`
      x. `$ which man`
      xi. `$ type man`
      xii. `$ whereis less`
      xiii. `$ help cd`
      xiv. `$ time sleep 2`

   (b) Use the appropriate man page, to check that you have interpreted the screen output correctly

6. *Creating New Files*

   (a) Try creating a new *empty* file in your home directory using the `touch` command, e.g.

   ```
   $ touch filename
   ```

   (b) Get the file details on *filename* using this command:

   ```
   $ ls -l filename
   ```

   (c) Wait 1 minute, then repeat the previous two steps, i.e.

   ```
   $ touch filename
   $ ls -l filename
   ```

      i. Which of the file details have changed?
      ii. What does this tell you about the purpose of `touch`? Check the man page if you are unsure.

   (d) Create a new files using re-direction

[5] Actually running this sort of search can take a long time, given that many systems contain over 500 man pages, some of which are very long.

[6] You should notice something very strange about the output from this string. The `cal` utility is perfectly functional, so what’s wrong?

   i. Create a new file containing the output from the `df` command, using re-direction, e.g.

      ```
      $ df > diskspace.txt
      ```

   ii. Ask a partner to create new files, with appropriate filenames, containing output from the commands used in the questions on “Finding Out About Your System and Users”.

7. *Appending information to files*

   (a) With a partner, choose several of the system information commands whose outputs may have changed since you completed the previous question. Practice appending the updated information to the file which contains the earlier output.

   (b) Create a file containing output from `w`, then append the output from `date` to it, i.e. time-stamp the output data.

8. *Using Simple Pipes*

   (a) Pipe the output from `who` through the `sort` command to reverse its order.

   (b) Sort your `/etc/passwd` file alphabetically and send the output to a new file (`passwd.sorted`).

   (c) Find out what `wc` does from its man page, then use it at the end of a pipe to analyse the output from other utlities.

   (d) Repeat the last step, limiting `wc` to counting words only

10.27 Overview Solutions

1. *Logging in*
   (a) N/A — it works or it doesn't.
   (b) N/A — it works or it doesn't.

2. *Changing password*
   (a) N/A — Responses will vary from system to system, depending on whether or not good password practice is enforced.

3. *Navigating Man Pages*
   (a) N/A
   (b) N/A — It is possible that some Linux distributions won't use `less` to display man pages. If that is the case, try to find out how you navigate under that setup and answer the same questions about it.
   (c) N/A
   (d) Keystrokes for basic man page navigation:

Instruction	Keystroke(s)
Top of man page	`g < ESC-<`
Bottom of man page	`G > ESC->`
Forward one screen	`f ^F ^V SPACE`
Backward one screen	`b ^B ESC-v`
Up one line	`y ^Y k ^K ^P`
Down one line	`e ^E j ^N RETURN`
pattern Search forward	`/`*pattern*
pattern Search backward	`?`*pattern*
Repeat *pattern* Search forward	`n`
Repeat *pattern* Search backward	`N`
Move to *n*th line	*n*`g`

Table 10.3: Keystrokes for basic man page navigation

N.B. Several different keystrokes can be used for the same movement. This is common in UNIX tools designed to operate from any keyboard. `less` always has a single key method. Multi-key methods are shown without spaces between them.

4. *Invoking the Right Man Pages*
   (a) i. `$ man -k whatis`
   or, slightly differently:
   `$ man -f whatis`
   ii. `$ man -K cdrom`
   iii. There is no easy way to do this yet. Later on you will learn about `grep` which will allow you to filter the output of `man -k print` to see only the information you require.
   (b) Practice using these flags to find and view man pages which deal with computer keywords your partner sets for you (and vice versa), e.g.
   i. e.g. `$ man -K jpg`
   ii. e.g. `$ man -K modem`

iii. e.g. `$ man -K NFS`

5. *Finding Out About Your System and Users*

(a) The listed command strings tell you about:

Command string	Output
`$ whoami`	Your username
`$ who am i`	Your username plus machine(s) and terminal you are on
`$ users`	Usernames of currently logged on users
`$ who`	Who is logged on, when and where
`$ w`	Who's logged on, when, where, what process and what system resources they are using
`$ date`	Current date and time, can set date/time
`$ cal 8 1999`	Calendar for August 1999
`$ cal 9 1752`	Calendar for September 1752. Strange because 12 days were 'lost' in the transition from Gregorian to Julian calendars
`$ df`	Disk free, i.e. summarises disk usage
`$ which man`	Full file and path name for the `man` executable file
`$ type man`	Much the same as `which man`
`$ whereis less`	Locates the `less` executable and its man page
`$ help cd`	Very brief help notes on the `cd` command. N.B. help only works on very few built-in commands
`$ time sleep 2`	The `sleep` command puts itself to sleep for 2 seconds. The `time` command then times the whole process and provides other data on the operation of the `sleep` command

Table 10.4: Output from basic system information commands

(b) See Table 10.4

6. *Creating new files*

(a) Your output should be something like:

```
$ touch filename.txt
```

(b) Your output should be something like:

```
$ ls -l filename.txt
-rw-rw-r--   1 davef  davef  0 Jul 21 17:59 filename.txt
```

(c) Your output should now be something like:

```
$ touch filename.txt
$ ls -l filename.txt
-rw-rw-r--   1 davef  davef  0 Jul 21 17:59 filename.txt
$ touch filename.txt
$ ls -l filename.txt
-rw-rw-r--   1 davef  davef  0 Jul 21 18:01 filename.txt
```

i. The time stamp has changed

ii. The real purpose of touch is to change time stamps, but it is handy for creating new empty files

(d) i. Reading *diskspace*.txt should produce something like this:

```
$ cat test.txt
Filesystem     Used Available Capacity Mounted on
/dev/hda1     65571   406394      14%   /
/dev/hdc1   5030416   650563      89%   /backup
/dev/hda5   2000097   857401      70%   /home
/dev/hda7     14289   457676       3%   /tmp
/dev/hda6   1136861   741727      61%   /usr
/dev/hdb     653004        0     100%   /mnt/cdrom
```

ii. N/A

7. *Appending information to files*

   (a) N/A

   (b) Your screen should look something like this:

```
$ w > test.txt
$ date >> test.txt
$ cat test.txt
  6:36pm  up 16 days, 23:07,  4 users,  load average: 1.03, 1.08, 1.02
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
davef    ttyp7    oakleigh:0.0      9:39am  0.00s  2.58s  0.02s  w
mikeb    ttyp4    kebab            Tue 3pm 23:07   0.20s  0.13s  -bash
davef    ttyp2    oakleigh          9:04am 15:34   7:24   0.07s  -bash
davef    ttypb    oakleigh          3:02pm  3:15m  3:00m  0.08s  -bash
Wed Jul 21 18:36:39 BST 1999
```

8. *Using Simple Pipes*

   (a) `$ who | sort -r`

   (b) `$ cat /etc/passwd | sort > passwd.sorted`

   (c) `wc` prints the number of lines, words, and bytes in files. To get these details for your `/etc/mime.types` file, you could do the following:

```
$ cat /etc/mime.types | wc
    275     488    7374
```

   i.e. 275 lines, 488 words, and 7373 bytes [7]

   (d) E.g.

```
$ cat /etc/mime.types | wc -w
    488
```

[7] N.B. `wc` only counts whitespace-separated words

Module 11

Printing Services

Objectives

On completing this module, you should be able to:

- Understand the basic principles of the Linux printing sub-system
- Use printing commands (`lp, lpr, lprm, etc`)
- Understand Samba printing
- Configure Samba to print from Windows hosts
- Configure Samba to print to Windows hosts

11.1 Linux Printing

- Completely network-oriented
- Any printer can be made available to any client (machine and application)
- All print jobs are sent to a queue
- Queues can be viewed, edited, maintained from anywhere
  - ♢ Subject to permission
- Formatted files can be sent straight to queues no

  i.e. no ‘device drivers’
- Printer configuration via text file `/etc/printcap`, see `man 5 printcap`

11.2 Printing documents

- Printing may be ‘dumb’
  - ◊ Data dumped straight to printer
- You get *BAD* results if formatting is wrong
- Your setup may be smart
  - ◊ Autodetect data formats and convert
- Older UNIX mainly dumb
- RedHat pretty smart - selects filters and transforms data streams if possible

11.3 Main Printing Tools

- `lpr` sends job to the queue for a named printer
- `lpq` returns info about jobs in a queue
- `lprm` removes unwanted jobs from a queue
- `lpc` enables system administrator to control the operation of the printing system
  - ♢ see `man lpc` for details
- Desktop environments may offer “drag ‘n’ drop”, visual facilities, etc

11.4 Using `lpr`

- Syntax:

```
lpr [options] file ...
```

- Main Options:

Flag	Options
`-P`	Name of the printer to send the job to
`-`*n*	Print *n* copies of the document
`-m`	Send mail on completion

Table 11.1: Main `lpr` options

- Example:

```
$ lpr -Pdjrmt -2 filetypes.txt
```

11.5 Using `lpq`

- Syntax:

```
lpq [options]
```

- Options:

Flag	Options
`-P`	Name of the printer/queue to interrogate
`-l`	Get info on each file within a job

Table 11.2: `lpq` options

- Example:

```
$ lpq -Pdjrmt -l
```

11.6 Using `lprm`

- Syntax:

```
lprm [options]
```

- Options:

Flag	Options
`-P`	Remove jobs from named printer/queue
`-`	Remove all jobs belonging to yourself
user	Remove all jobs belonging to *user*
n	Remove job number *n*

Table 11.3: `lprm` options

- Example:

```
$ lprm -Pdjrmt -davef
```

11.7 Samba Printing

- To configure a Windows machine to use a Linux printer:
  - ♢ Locate the printer on your network, by browsing neighbourhood or directly naming
  - ♢ Connect the printer to your system, as with windows network printers
- Sending a document from a windows host to a Linux printer is just as simple:
  - ♢ Open the print dialogue box in the appropriate application
  - ♢ Ensure the Linux printer is selected for use
  - ♢ Choose available printing options
  - ♢ Press enter
- All exactly as if you were using a Windows printer

11.8 Printing to a Windows Printer via Samba

- There are only two good reasons for this:
  - ♢ You have a Windows-only printer
  - ♢ You can't think of another use for your Windows PC
- Requires the following steps:
  1. Create a Linux printer in `/etc/printcap`, e.g.
     ```
     smb:lp=/dev/null:sd=/var/spool/smb:sh:if=/usr/local/samba/smbprint
     ```
  2. Copy sample smbprint filter to appropriate place, e.g. from:
     `/usr/doc/samba-xxx/examples/printing/smbprint`
     to:
     `/usr/local/samba/smbprint`
  3. Create a spool directory for each Windows printer, with right permissions, e.g.
     `/var/spool/lpd/`*winprn1*
  4. ID each Windows printer in a config file, e.g.
     `/var/spool/lpd/`*winprn1*`/.config`
  5. Put something like this in each config file:
     ```
     server=PC_SERVER
     service=PR_SHARENAME
     password="password"
     ```
     e.g.
     ```
     server=JULIES_PC
     service=WINPRN1
     password="******"
     ```

Module 12

Basic Shell

Objectives

On completion of this module, you should be able to understand and use the Linux shell to create and combine tools.

Topics covered include:

- An overview of the command line
- The software tools model
- File names and types
- Shell programming
- Command scripts
- Job control
- I/O - pipes and redirection

12.1 Introduction

- The standard command line interpreter under Linux is `bash` (`/bin/bash` or `/bin/sh`)
- An enhanced version of the classic Bourne shell
- Shares most features of other shells (`C`, `Korn`, etc) and has some more advanced features
  - ♢ 'Plumbing' - transparent redirection and pipes
  - ♢ Background processes
  - ♢ Process suspension, resumption, termination
  - ♢ Filename completion and wildcard generation
  - ♢ History

12.2 Getting around the command line

- You can use the cursor keys to move around and edit the current line[1]
- By default, `bash` uses `emacs`-like keystrokes for navigation and editing. Here are 4 examples:

Keystroke	Action
`^a`	Move to the beginning of the line
`^e`	Move to the end of the line
`^k`	Delete to the end of the line
`^w`	Delete the previous 'word'

- To choose `emacs` or `vi`-like keystrokes:

```
$ set -o emacs
$ set -o vi
```

- `bash` man page gives details of all keystrokes

[1]This may not work on badly-configured systems

12.3 History

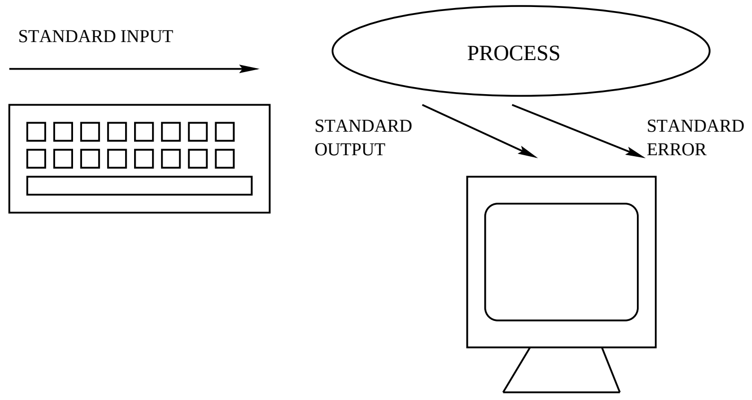
- Bash remembers used commands (in a ‘history’)
- Old commands are retrievable in different ways
- Repeat the previous command by typing `!!`
- Execute the *n*th previous command by typing `!-`*n*
- Typing `!string` repeats the last command beginning with `string`
- To view your history command by command, use the `up` and `down` cursor keys
- View your history at any time by typing `history`
- History is a *very* useful feature, if used well
  - ♢ Incrementally searchable using `CTRL-R`

12.4 Plumbing

- Processes typically start with three files open:

Name	Descriptor
Standard input	0
Standard output	1
Standard error	2

- Later we see how to refer to their *file descriptors*
- These are normally connected to the keyboard and your command-line terminal



12.5 Plumbing (continued)

- Data can be redirected by the shell
  - ◇ Transparently to the process concerned
  - ◇ Any or all streams can be redirected
  - ◇ You can redirect to/from a file or to/from another process
- Redirection to a process is known as ‘piping’

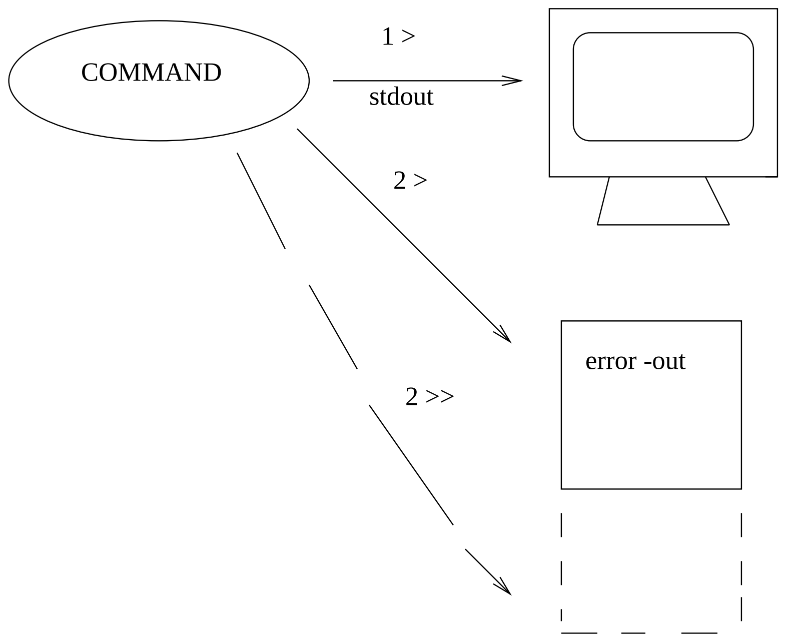
12.6 Output Redirection

- Redirection of output is done using '>'
- For example:

  ```
  $ command > output
  ```

  Creates the file `output` (or overwrites it if it already exists) and places the standard output from `command` into it
- We can append to a file rather than overwriting it by using >>
- > and >> are actually shorthands for `1>` and `1>>`
- Error output can be redirected using `2>` or `2>>`

  ```
  $ command 2> error-out
  ```

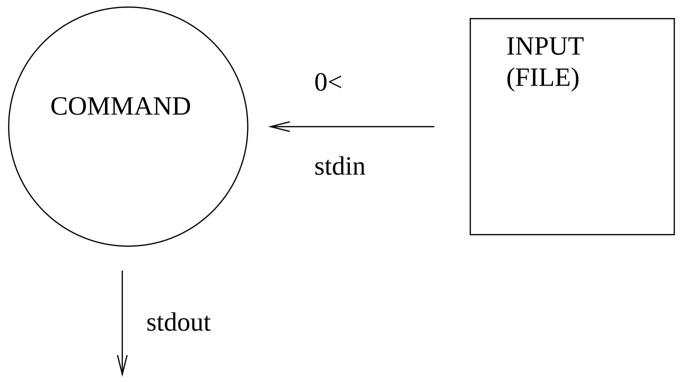


12.7 Input Redirection

- `<` redirects standard input from a file, e.g.

```
$ command < input
```

- `command` will now take the contents of the file `input` as its input
- This could also be written as `0<`
- Consistent with `>` and `1>`



12.8 Combining Redirection

- Redirect more than one descriptor by giving more than one redirection, e.g.

```
$ command 1> output 2> error
```

- Group redirections using the `>&` operator, e.g.

```
$ command 1> output 2>&1
```

  - ♢ Output to the file called `output` (`> output`)
  - ♢ Send errors to the same place as the standard output (`2>&1`)

- The order of these is *very* important
- The redirections are evaluated left-to-right, e.g. the following differs from the previous example

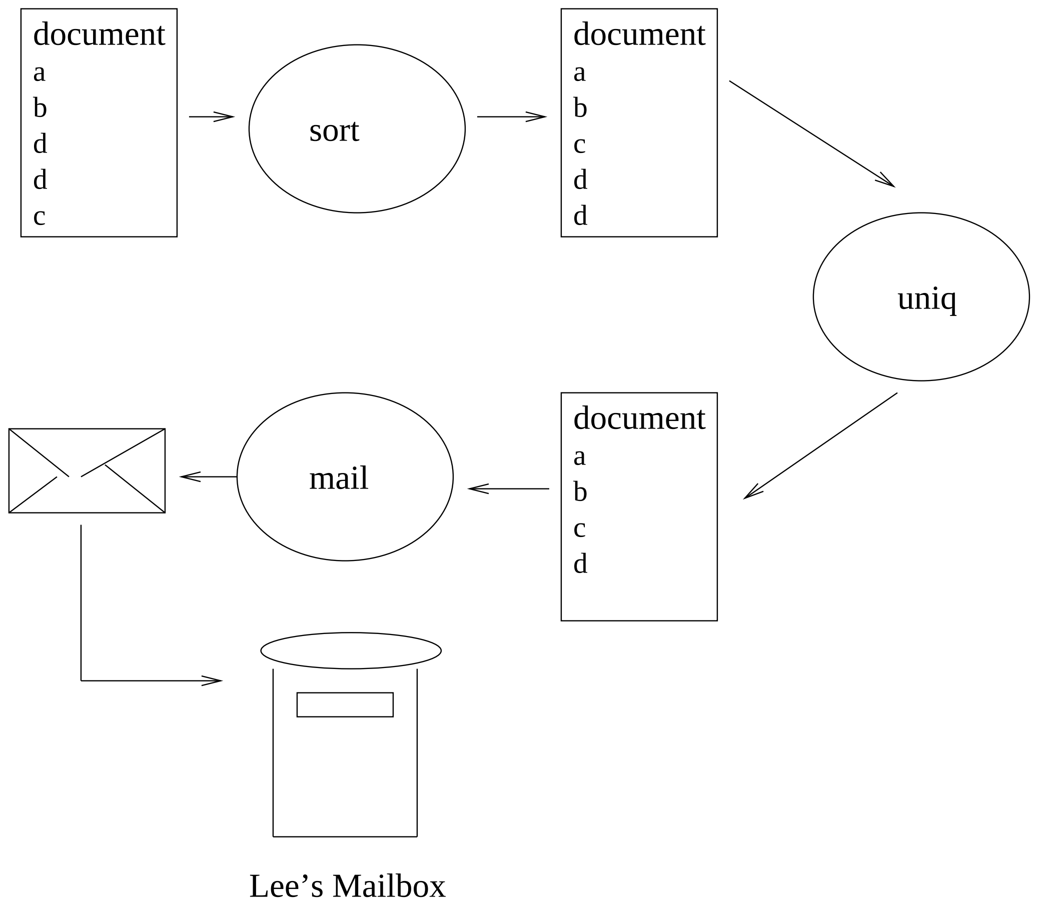
```
$ command 2>&1 > output
```

  - ♢ It sends error to the normal output and normal output to the file called `output`

12.9 Pipelines

- You can output to another process with '`|`'
  - ♢ Known as the *pipe* symbol
- A *pipe* connects the output of one process to the input of another
- The data waiting to be transferred is buffered
- The processes run concurrently
- Linux ensures that the processes keep in step
- For example:

```
$ sort document | uniq | mail lee
```



12.10 Background Processes

- Most commands run to completion before you get your shell prompt back
- A ‘background’ process continues while you get your prompt back immediately
- To launch a process in the background place `&` at the end of the line, e.g.

```
$ sort /var/log/maillog > output &
```

- Unless you use redirection (plumbing), output and error continue to appear on your terminal
- Input is disconnected, so typing goes to the shell, not to the background process
- If a process needs user input, and can’t take it from a file, it is ’stopped’
  - ◇ It won’t resume until brought to the foreground to receive input
- You should normally start background processes with their output and error redirected to a file, e.g.

```
$ sort big_file > output 2> error_output &
```

12.11 Background Processes (continued)

- Running processes can be put in the background
  - ♢ Suspend the process by typing `^Z` in the terminal that the process is running in
  - ♢ Put the process in the background using `bg`
- Bring a process back to the foreground using `fg`
- `fg` and `bg` operate on the most recent process by default
  - ♢ Change to a process, by job number or name
- `jobs` displays current shell processes:

```
$ jobs
[1]+  Stopped (tty output)    top
$ fg %1
```

12.12 Background Processes and `nohup`

- Sometimes it is necessary to start a process and leave it running when you log out
- If your shell is killed, any background processes will also be lost
- `nohup` gets round this by detaching the process from the terminal
- Always redirect output and error with `nohup`, e.g.

```
$ nohup sort bigfile > out 2>&1 err.out
```

- If you don't redirect them then they will end up in `./nohup.out` and `./nohup.err`

12.13 Command Grouping and Sub-shells

- `bash` can execute multiple commands on a line
- Sequential commands are separated by '`;`'

```
$ sort data ; mail lee < sorted_data
```

- It's possible to launch a sub-shell to execute a command or group of commands
  - ♢ Put commands in parentheses, e.g.

    ```
    $ (command1 ; command2)
    ```

- Can also put a subshell in background, e.g.

```
$ (command1 ; command2) &
```

12.14 Process Management

- `ps` (process status) prints info about a user's processes :

```
PID   TTY STAT TIME COMMAND
22074 p0  S    0:02 Eterm -t trans
22075 p0  S    2:13 emacs -bg black
22081 p0  S    0:00 asclock
22590 p5  R    0:00 ps
```

- `jobs` only prints info about processes belonging to the current shell
- `wait` postpones shell until process is finished
  - ♢ Usually given a process id as an argument
  - ♢ If no argument is given it waits until all the shell's processes have terminated
- `kill` is used to send *signals* to processes
  - ♢ Can terminate background processes
- Some processes use signals to trigger tasks, e.g. log rotation, re-reading config files, etc

12.15 Signals

- `kill` can be given a signal name or number
- There are a variety of signals :

SIGHUP	1	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Interrupt from keyboard
SIGQUIT	3	Quit from keyboard
SIGKILL	9	Kill signal
SIGTERM	15	Termination signal
SIGUSR1	30 10 16	User-defined signal 1
SIGUSR2	31 12 17	User-defined signal 2

12.16 Signals (continued)

- Unless specified, `kill` sends a SIGTERM which causes most processes to terminate
- If a process is unresponsive, it can be forcibly killed by sending it SIGKILL

```
$ kill -9 1512
1512: Terminated
```

  or

```
$ kill -KILL 1512
1512: Terminated
```

- Can only signal your own processes
  - ♢ Superuser can signal all

12.17 Background Processes: `top`

- `top` displays the processes running on a machine
- Results can be sorted in various ways
- Options:
  - ♢ See `man top` for full details, including command-line options
  - ♢ Inside `top` use `h` for help on interactive options
- Typical output:

```
  PID USER    PRI  NI  SIZE  RSS SHARE STAT LIB %CPU %MEM   TIME COMMAND
22594 user     10   0   736  736   556 R      0  8.2  0.5   0:00 top
    1 root      0   0   144   96    76 S      0  0.0  0.0   0:03 init
    2 root      0   0     0    0     0 SW     0  0.0  0.0   0:19 kflushd
    3 root    -12 -12     0    0     0 SW<    0  0.0  0.0   2:42 kswapd
  486 root      5   5  3160 2356   804 S N    0  0.0  1.8   0:03 mysqld
  869 root      0   0    68   12    12 S      0  0.0  0.0   0:00 mingetty
  838 www       0   0 11468 6280   488 S      0  0.0  4.9   4:14 squid
   48 root      0   0   100   80    48 S      0  0.0  0.0   0:01 kerneld
  230 root      0   0   384  372   272 S      0  0.0  0.2   0:12 syslogd
  239 root      0   0   164  120    72 S      0  0.0  0.0   0:00 klogd
  250 daemon    0   0   164  132    88 S      0  0.0  0.1   0:00 atd
  261 root      0   0   192  160   112 S      0  0.0  0.1   0:01 crond
  272 bin       0   0   244  224   168 S      0  0.0  0.1   0:00 portmap
  283 root      0   0   572  296   248 S      0  0.0  0.2   0:17 snmpd
  295 root      1   0   136   88    60 S      0  0.0  0.0   0:00 inetd
  306 root      0   0   516  488   224 S      0  0.0  0.3   0:06 named
  317 root      0   0   124   56    48 S      0  0.0  0.0   0:00 lpd
```

- N.B. `top` is not available on all unices

12.18 Filename Generation

- Some characters are ‘special’ to the shell

Chars	Meaning
`*`	Matches any string, including the null string
`?`	Matches any single character
`[...]`	Matches any one of the enclosed characters. A pair of characters separated by a minus sign denotes a range. Any character lexically between those two characters, inclusive, is matched. If the first character following the [is a ! or a ^ then any character not enclosed is matched. A - or] may be matched by including it as the first or last character in the set.

Table 12.1: Special characters under `bash`

- Special characters can be used to match filenames, e.g. to show files beginning with `f`

```
$ echo f*
```

- To show files starting with `f`, followed by a vowel:

```
$ echo f[aeiou]*
```

12.19 Quoting Mechanisms

- Sometimes it's necessary to ignore a character's special meaning
- Use a backslash (\) to quote a special character, e.g. to list a file called `f*`

```
$ ls f\*
```

- To quote a longer string, enclose it in quotes :

'	disable all interpretation
"	disable filename generation and blank space interpretation

12.20 Shell built-in commands

- Some commands must be built in to the shell, because they can't be executed independently
  - ◇ `cd`, if executed independently would change its own directory, *not* that of your shell
  - ◇ `umask`, would change its umask, *not* the shell's
  - ◇ `logout`
  - ◇ `history`
- Other commands are built in for speed e.g.
  - ◇ `pwd`
  - ◇ `echo`

12.21 Basic Shell Exercises

1. *Redirection*

   (a) Try typing the following commands exactly as they appear here:

   ```
   $ cat > newfile 2> newfile.error
   $ car > newfile 2> newfile.error
   $ cat > newfile 2>&1
   $ car > newfile 2>&1
   $ cat < newfile
   $ echo foo | cat > newfile 2>&1
   $ ech foo | cat > newfile 2>&1
   $ echo foo | car > newfile 2>&1
   $ (ech foo | cat) > newfile 2>&1
   ```

   Make sure you understand what happens in each case, ask the tutor if you are note sure.

2. *Filename expansion and Quoting*

   (a) Do the following in the `/bin` directory

      i. List all filenames with exactly three characters.
      ii. List all filenames with exactly three characters in which the second character is a vowel.
      iii. List all filenames with a, b, c, or d as the last character.
      iv. Construct a command to print the number of filenames consisting of exactly three characters.
      v. Construct a command to print the total number of files with exactly two, three or four characters in their name. (You may find the wc utility useful here, check `man wc` for more information.)

   (b) Compare the effect of the following commands:

   ```
   echo $HOME
   echo "$HOME"
   echo '$HOME'
   echo *
   echo "*"
   echo '*'
   echo $HOME/*
   echo "$HOME/*"
   echo '$HOME/*'
   ```

   (c) Change back to your home directory and try to create a file with the name *. Was this a sensible thing to do? How would you delete it? (Be *very* careful!)

   (d) Create a file called `-file`. Try to remove this. Use the `rm man` page to help you.

3. *Background processes and* `nohup`

   (a) Start the command `sort /dev/random` in the background in your current shell

   (b) Bring it back to the foreground and terminate it by typing ^C

   (c) Start it again, and once more so that you have two copies running in the background

   (d) Bring them to the foreground and terminate them in the order you started them
   (e) Start the same command in the background, and terminate it using `kill`

4. *Grouped commands*

   Compare the following command sequences, and make sure you understand the differences :

   (a)
   ```
   cd /tmp
   cd /usr; ls
   pwd
   ```
   (b)
   ```
   cd /tmp
   (cd /usr; ls)
   pwd
   ```
   (c) `sleep 5; sleep 5 &`
   (d) `(sleep 5; sleep 5) &`

   Check you can use your history to get at and repeat any of the commands you have typed.

12.22 Basic Shell Solutions

1. N/A
2. *Filename expansion and Quoting*
   (a) These are solutions which will do the job, there may be other ways of acheiving the same thing
      i. ls ???
      ii. ls ?[aeiou]?
      iii. ls *[abcd]
      iv. ls ??? | wc -l
      v. (ls ?? ; ls ??? ; ls ????) | wc -l

   (b) When not quoted $HOME gives the name of your home directory. This is variable substitution. We can see that this substition still happens inside " quotes, but not inside ' quotes. `echo *` expands to all the filenames in the current directory. This is filename generation and doesn't happen in either quotes.

   (c) It is not a wise choice to name anything with a filename containing special characters.

   You can delete the file safely using either:
   - `rm "*"`
   - `rm '*'`

   (d) You can delete the file by using `rm - -file`

3. *Background processes and* `nohup`

   (a)
```
$ sort /dev/random &
```
   (b)
```
$ fg
^C
```
   (c)
```
$ sort /dev/random &
[1] 26409
$ sort /dev/random &
[2] 26410
```
   (d)
```
$ jobs
[1]-  Running                 sort /dev/random &
[2]+  Running                 sort /dev/random &
$ fg %1
sort /dev/random
^C
$ fg %2
sort /dev/random
^C
$
```
   (e)
```
$ sort /dev/random &
[1] 26462
$ kill -9 26462
```

4. N/A

Module 13

Shell Programming

Objectives

By the end of this session you should know about:

- Shell variables and their use
- Conditionals and flow-control
- Basic scripting techniques
- Use of parameters
- The use of *here* documents
- Command substitution
- Use of tools like `find` and `expr`

13.1 Introduction

- The shell is not only a *command interpreter*
- It can also be used as a *programming language*
- Shell programs are often called *shell scripts*
  - ♢ Or simply – *scripts*
- By the end of the section, you should understand how to:
  - ♢ Use the shell as a programming tool

13.2 Writing and Running Shell Scripts

- The standard Unix shell is the *Bourne Shell*[1]
- Linux provides an advanced clone -
  ‘Bourne-Again shell’ (*bash*)
- Just like any other process
- It can execute subshells and have its input and output redirected
- You can put commands in a file and get the shell to read from that file
- Running a shell program:

1. `$ sh filename`

2.
   ```
   $ chmod +x filename
   $ ./filename
   ```

[1]Named after its author, Steve Bourne

13.3 Subshell or Subroutine?

- The previous examples launch a *subshell* to execute the code
- Changes to the process environment exist only in that subshell (e.g. ID of current directory)
- You can run a shell program as a subroutine of your current shell: `$ .  filename`
- This is the *only* way to change the environment of the current shell
- If the subroutine changes the current directory or other shell variables you will still see the effect after it has finished running

13.4 Processing Commands

- The shell reads its input
- Input is split into command, arguments, and ‘plumbing’ (I/O re-direction)
- It performs
  - ♢ Command substitution
  - ♢ Variable and parameter substitution
  - ♢ Blank space interpretation
  - ♢ Filename generation
  - ♢ ‘Plumbing’
- You can see what it is doing by using `set -x`, e.g.

```
$ set -x
$ ./filename
```

13.5 Command Substitution

- Used as part of a command line
- The standard output of a command is substituted into the command line
- Newlines become spaces, e.g.

  ```
  foo
  bar
  ```

  becomes

  ```
  foo bar
  ```

- Do this by using back-quotes (‘).
- Examples:

  ```
  $ echo today it is ‘date‘
  today it is Thu May 20 15:42:58 BST 1999
  $ echo ‘ls‘
  axhome bin etc info mail man systemprog
  $ echo I am in ‘pwd‘
  I am in /home/lee
  ```

13.6 Shell Variables and Variable Substitution

- Variables are items whose content (i.e. ‘value’) *varies* but are identified by a *constant name*
- The shell is a string-based language
- Variable assignment looks like `varname=string`
- The value of `varname` is given by `${varname}` or `$varname`
  - ♢ N.B. Variable names are case-sensitive
- Use `${varname}` when the value is concatenated (joined) with other characters, e.g.

```
$ myname=Lee
$ echo $myname
Lee
$ echo ${myname}abc
Leeabc
```

- Using a variable before it is set gives a null string

```
$ echo $mynameabc

$
```

13.7 Shell Environment Variables

- *Environment variables* are shell variables whose values are *exported*

  i.e. passed *down* to all child processes of the current shell

- This is done by using the `export` command
- Changes don't pass back up to a parent process
- See what is currently exported using `set`
- Some variable names have built-in meanings:

VARIABLE	MEANING
`PATH`	A list of directories searched to find executable commands: `PATH=:/home/lee/bin:/bin`
`HOME`	The user's home directory
`MAIL`	Pathname of received mail
`IFS`	Inter-field separator, characters used to split a command line into words
`PS1`	The shell's usual prompt

`PS2`	Prompt given on subsequent lines of a multi-line command.
	`$ echo hello there \`
	`> lee`
	`hello there lee`
	`$`

13.8 Examples

```
$ linux=free
$ echo $linux
free
$ sh
$ echo $linux

$ exit
$ export linux
$ sh
$ echo $linux
free
$ linux=unix
$ export linux
$ echo $linux
unix
$ exit
$ echo $linux
free
```

13.9 Examples (continued)

```
$ set
APPLIXPREFIX=/backup/lib
LD_LIBRARY_PATH=/home/lee/lib:/usr/local/lib
LOGNAME=lee
LS_COLORS=ln=32:di=35:pi=30:ex=31
MAIL=/var/spool/mail/lee
OSTYPE=Linux
PATH=/home/lee/bin:/bin:/usr/bin:/usr/local/bin
PPID=57
PS1=\u @ \t \w
PS2=>
PS4=+
PWD=/home/lee
SHELL=/bin/bash
SHLVL=3
TERM=xterm
linux=free
```

13.10 `set`

- `set`, on its own, lists the values of environment variables
- Inside a shell program, `set`, followed by a list of arguments places them in the variable called `$@`
- Example:

```
$ uptime
 11:07am up 78 days, 17:58, 7 users, ...
$ set ‘uptime‘
$ echo $1
11:07am
$ echo $3
78
$ echo $2
up
```

13.11 Quoting

- Characters with special meanings to the shell:

  `< > & $ | * ? ' " ‘ # [ ] ( ) ; { } \`

- Special meanings can be ignored by *quoting*
- Single characters are quoted with a preceding `\`
- Strings are quoted by surrounding `'` or `"`
- N.B. There are two *different* quotes used by the shell; both have distinct meanings:

`"..."`	Don’t expand shell special characters, do perform variable, command and parameter substitution
`'...'`	Don’t expand shell special characters or perform substitution

- N.B. Don’t confuse apostrophes with back ticks — they are used for command substitution

13.12 Examples

```
$ export linux=free
$ echo $linux
free
$ echo "$linux"
free
$ echo '$linux'
$linux
```

```
$ echo this_is_the_file_test > test me
$ ls -l
total 1
-rw-rw-r-- 1 lee      lee      5 Jun  1 13:47 test
$ echo this_is_test_me > "test me"
$ ls -l
total 2
-rw-rw-r-- 1 lee      lee      5 Jun  1 13:47 test
-rw-rw-r-- 1 lee      lee      5 Jun  1 13:47 test me
$ echo this_is_test_me_too > 'test me'
$ ls -l
total 2
-rw-rw-r-- 1 lee      lee      5 Jun  1 13:47 test
-rw-rw-r-- 1 lee      lee      6 Jun  1 13:48 test me
$ cat "test me"
this_is_the_file_test
$ cat test*
this_is_the_file_test
this_is_the_file_test_me_too
```

13.13 `.profile` File

- Each time you log in the Bourne shell executes the file called `.profile` in your home directory
- It is executed as a subroutine, so it may alter the environment of your current shell
- You will typically use it to set your *preferences* - your PATH variable is a classic, as is PS1 [2]
- The files `.bashrc`, `bash_profile`, and/or `.bash_login` are also read sometimes, according to context (See over)

[2]What your shell prompt looks like

13.14 `.profile` (continued)

- The logic for determining which files are run is as follows (taken from `man bash`)

  1. Login shells:

     ```
     On login (subject to the -noprofile option):
       if /etc/profile exists, source it.
       if ~/.bash_profile exists, source it,
         else if ~/.bash_login exists, source it,
           else if ~/.profile exists, source it.
     On exit:
       if ~/.bash_logout exists, source it.
     ```

  2. Non-login interactive shells:

     ```
     On startup (subject to the -norc and
                 -rcfile options):
           if ~/.bashrc exists, source it.
     ```

  3. Non-interactive shells:

     ```
     On startup:
       if the environment variable ENV is non-null,
       expand it and source the file it names, as
       if the command
           if [ "$ENV" ]; then . $ENV; fi
       had been executed, but do not use PATH to
       search for the pathname.  When not started in
       Posix mode, bash looks for BASH_ENV before ENV.
     ```

- ‘Source’ is a synonym for ‘execute as a subroutine’

13.15 Arguments

- You can pass command line arguments to a shell by using either:

  ```
  $ sh filename arg1 arg2 ...
  ```

  or:

  ```
  $ ./filename arg1 arg2 ...
  ```

- Arguments 1 to 9 are seen as `$1 $2 ... $9`
- `$0` is the name the program was called with (*filename* in the examples above)
- Later we will see the `shift` command which gives access to arguments beyond 9

13.16 Shell Parameters

- Shell parameters are *read-only* variables, i.e. don't normally change
- Hold information about the status of the shell
- Examples:

Shell Parameter	Meaning
`$?`	Exit Status of last executed command
`$#`	Number of command line arguments available
`$*`	A string containing all of the command line arguments
`$@`	A string containing all of the command line arguments
`$$`	The process-id of the current shell
`$!`	The process-id of the most recent background process

- `$*` and `$@` differ when quoted. Check `man bash`

13.17 Blank Interpretation

- Command line is scanned for internal field separators (defined by IFS variable)
- Command line is split into *command* and *arguments*
- Explicit empty arguments such as `""` or `’’` are preserved
- Null arguments resulting from command substitution are discarded
- If you want to preserve null arguments resulting from command substitution:
  - ♢ Concatenate with an explicit empty argument: `’’‘which foo‘`
- Implicit null arguments (e.g. substituting the value of an unset variable) are removed

13.18 Comments

- `bash` has one comment character, the #
- A comment starts with the hash character and ends at the end of that line
- Here is an example:

```
# Program to output a calendar for this month
cal ‘date +"%m %Y"‘ # This is also a comment
```

- N.B. Make *meaningful* comments

13.19 `read`

- The `read` command is another way of assigning values to variables
- Reads the next line of text entered from *stdin*
- When it is asked to read more than one variable, (e.g. read *var1 var2 var3*) the first word from the input is assigned to the first variable, the second word to the second variable etc, and the rest of the line to the last variable
- Example:

```
$ read line
is it only me?
$ echo $line
is it only me?
$ read first second rest
is it only me?
$ echo $first
is
$ echo $second
it
$ echo $rest
only me?
```

13.20 `shift`

- The `shift` command lets you access arguments above $9 by *shifting* them down one place

  i.e. the theoretical $10 becomes $9

- `$0` always stays the same, because it is the program
- If there is no argument to shift down the $9 variable becomes empty
- `shift n` is equivalent to executing `shift` `n` times
- Example:

```
$ set ‘date‘
$ echo $*
Tue Jun 1 15:23:04 BST 1999
$ echo $1
Tue
$ shift
$ echo $1
Jun
$ shift
$ echo $1
1
$ shift 3
$ echo $1
1999
$
```

13.21 Exit Status or Return Code

- Almost all commands give an exit status
- Identifies whether the command had any problems running
- `bash` can test these return codes
    - ♢ A value of zero is taken to mean success (or *true*)
    - ♢ Non-zero return codes signify failure (*false*)
- The exit status of the last executed command is stored in the shell parameter $?
- Shell programs can set their return status using the `exit` command
    - ♢ This terminates the program and gives the designated value as the return code, e.g.

      ```
      exit 0
      ```

13.22 `test`

- `test` *`comparison`* indicates whether a comparison was succesful or not
- It can test a number of different things :
  - ♢ File status
  - ♢ Numerical comparisons
  - ♢ String comparisons
- Used with the flow-control constructs such as `if` and `while` (explained later)

13.23 File Status Tests

- Used to check a file for certain properties

Command	Meaning
`test -f file`	true if `file` exists
`test -r file`	true if `file` is readable
`test -w file`	true if `file` is writeable
`test -x file`	true if `file` is executable
`test -u file`	true if `file` is set-user-id
`test -g file`	true if `file` is set-group-id
`test -k file`	true if `file` has sticky bit set
`test -d file`	true if `file` is a directory
`test -b file`	true if `file` is a block special
`test -c file`	true if `file` is a character special
`test -p file`	true if `file` is a name pipe (*fifo*)
`test -s file`	true if `file` has something in it
`test -t filedes`	true if descriptor `filedes` is a terminal (If fildes is not given then this checks standard output)

13.24 Relational Tests

- Numerical tests

`test nl -op n2`	true if the relationship `op` is valid for the numbers `n1` and `n2`
relationships	`-eq, -ge, -gt, -le, -lt, -ne`

- String tests

`test s`	true if `s` is non-null
`test -z s`	true if `s` is zero length
`test -n s`	true if length of `s` is not zero
`test s1 = s2`	true if `s1` and `s2` are identical
`test s1 != s2`	true if `s1` and `s2` are not equal

- The test

```
test $x = $y
```

will fail with an error message if either `x` or `y` are uninitialised.

13.25 `for` Loop

- Executes commands once for each word in a list
- Each time round the index variable is assigned the value of the next word
- Syntax is:

```
for index in word1 word2 ... wordn
do
  commands
done
```

- If we want to do something for each command line argument we can do

```
for index in "$@"
```

or

```
for index
```

- Filename, variable and command expansion are performed on the list, e.g.

```
for files in f*
```

would assign the names of all files beginning with an `f` to the variable `files` one by one

13.26 `if`

- Three basic forms:

1.
```
if ...
  then
    ...
fi
```

2.
```
if ...
  then
    ...
  else
    ...
fi
```

3.
```
if ...
  then
    ...
  elif ...
    then
      ...
  else
    ...
fi
```

13.27 `if` (continued)

- Executes the test and takes the appropriate action
- `elif` is short for "else if"
- There can be any number of `elif` clauses
- Example:

```
for file in *
do
  if test -d $file
    then echo $file: is a directory
  elif test -x $file
    then echo $file: is executable
  elif test -w $file
    then echo $file: is writable
  else
    echo $file: is a standard file
  fi
done
```

- Everything following an `if` or `elif` to the end of the line or a `;` is executed as the test

13.28 `true` and `false`

- Return the appropriate exit status
- Generally used for infinite loops ("`while true ...`")
- `true` returns 0
- `false` returns some non zero value, typically 1 but could be anything non-zero!

13.29 `while` and `until`

- The `while` command repeats a group of commands as long as the test remains true
- The `until` command repeats a group while the test remains false
- Example:

```
until who | grep lee > /dev/null
do
  sleep 15
done
echo Hi Lee | mail -s "Greetings!" lee
```

13.30 `break` and `continue`

- Used to alter the flow of loop structures
- `break` breaks out of the immediately-enclosing loop
- `break n` breaks out of the `n`th enclosing loop
- `continue` jumps back to the beginning of the enclosing loop
- `continue n` jumps to the beginning of the `n`th enclosing loop
- For example

```
file=$1
for i in *
do
  echo -n "Checking $i"
  if test $1 = $i; then
    echo " - Found!"
    break
  else
    echo " - Nope."
    continue
  fi
  echo "Never get here ..."
done
```

13.31 `case`

- The `case` statement enables multiple comparisons based on pattern-matching
- A neater form of

```
if ...
  then
    ...
  elif
    ...
  elif
    ...
fi
```

- Structure:

```
case test-string in
  match1)      commands ;;
  match2)      commands ;;
esac
```

- Two semi-colons act as a single symbol in this context

13.32 `case` (continued)

- The `case` statement uses the same pattern matching as filename generation
- Patterns are scanned in lexical order
- Example:

```
echo -n "Enter a digit or letter : "
while true
do
  read answer
  case "$answer" in
    [0-9])
      echo "You entered the digit $answer."
      ;;
    [a-zA-Z])
      echo "You entered the letter $answer."
      ;;
    *)
      echo "Invalid input!"
      ;;
  esac
done
```

13.33 `case` (continued)

- Alternative matches are specified using the or character |
- The previous example could have had:

```
case "$answer" in
  0|1|3|5|6|9)
    echo "You entered the digit $answer."
    ;;
```

- Only the first matching pattern will be used:
- We only reach the invalid input line when the input doesn't match either of the first two options

13.34 Simple Conditionals

- The example in Section 13.32 provides a shorter form of the `if` construct
- The line

  ```
  read answer || break
  ```

  means `break` if the command to the left of the `||` evaluates to *false*
- This is known as the `OR` operator
- The second short from is the `AND` operator `&&`

  ```
  read answer && break
  ```

- The second command is executed only if the first returns `true`

13.35 Here Documents

- Here documents are ‘virtual input files’
- A portion of the script file is sent to a command as if it were that command’s normal `stdin`
- Input is taken from the following text until the end delimiter is encountered
- Example

```
cat <<EOF > test.output
This
is
a
test
EOF
```

- The above is equivalent to typing `cat > test.output` on the command line then entering :

```
This
is
a
test
^D
```

13.36 Use of commands

- Can run anything you want from a script
- Common untilities in scripts are :
  - `find`
  - `expr`
  - `tar`
  - `mail`

13.37 `find`

- `find` searches the filesystem in real time; making disks work hard
- Can find files by name, type, size, dates, e.g
  - ♢ To find all files ending with `.jpg` under the current directory:

    ```
    find .  -name "*.jpg"
    ```
  - ♢ To find all filenames ending in `.jpg` *and* modified in the last 8 days below `/etc`

    ```
    find /etc -name "*.jpg" -mtime -8
    ```
- Tests can be combined with `-o` and negated with `!`, for example:
  - ♢ To find all filenames *not* ending in `.jpg` *or* modified in the last 8 days

    ```
    find .  !  -name "*.jpg" -o -mtime -8
    ```
- Can execute commands on the files it finds. The name of the file found is placed in {}

  ```
  find . -name "*.gif" -exec ls -l {} \;
  ```
- N.B. The latter spawns separate processes for each find

13.38 Evaluate Expressions (`expr`)

- Evaluation works out the result of some expression

  i.e. computes the value of the expression
- `expr` is used to evaluate expressions
- Takes arguments and operators
- Prints the result
- Returns zero or non zero depending on the result
  - ◇ Scripts often test this result and act accordingly
- Watch out for special meaning to shell of characters like * and < or >, e.g..

```
$ expr 1 + 2
3
$ expr 6 \* 7 #must escape the '*'
42
$ if expr 1 + 2 != 3 > /dev/null
> then
> echo yes
> else
> echo no
> fi
no
```

13.39 `expr` (continued)

- Often used for looping in scripts

```
i=0
while expr $i != 20 >/dev/null
do echo $i
  i=`expr $i + 1`
done
```

- Has some string manipulation facilities too

```
$ expr index abcdefg d
4
$ expr substr abcdefghijk 3 3
cde
```

13.40 Summary

- In this section we have covered the basics of how to create and use shell scripts.
  - ♢ Substituting commands with back ticks.
  - ♢ Shell variables and parameters.
  - ♢ The `shift` command.
  - ♢ The exit status of commands.
  - ♢ The flow control constructs, `if`, `while`, `until` and `case`.
  - ♢ The `test` command.
  - ♢ *Here* documents.

13.41 Shell Programming Exercises

1. (a) Write a shell script, which prints the number of users currently logged in.
   (b) Construct a script, which takes a username as an argument. It should print all lines of output from the `who` command which concern that user.
   (c) Edit your `.profile` to
   - Print a personalised greeting when you log on
   - Print the date
   - Personalise your prompt
   - Show how many mail messages are in your mailbox
     - ♢ You should note that your mailbox is typically stored in /var/spool/mail/username, and that messages are delimited by lines that begin "From "

2. (a) Try typing several of the Linux commands you have learnt, followed by

```
echo $?
```

   For example you could try :

```
grep root /etc/passwd
echo $?
grep nonuser /etc/passwd
echo $?
grep root /etc/paswd
echo $?
```

   (b) Write a shell script `arguments` which when run like this
   `arguments one two three four`
   gives out

```
There are 4 arguments
one
two
three
four
```

   (c) What would you expect if you typed
   `arguments one two 'three four'`
   `arguments one ' ' three`
   or
   `arguments a*`
   (d) Write a shell script to print out all 100 numbers from 00 to 99.

3. (a) Write a shell procedure, `save`, which copies the files specified as arguments into the directory `$HOME/backup`. Your procedure should create the directory `$HOME/backup` if it does not already exist.
   (b) Write a shell procedure, `whoare`, which takes a number of usernames as arguments. For each name, a message should be printed showing whether the user is logged in, exists in the password file, or does not exist

4. (a) Write a shell procedure, `mydate`, with the following specification:

```
mydate [ -d ] [ -t ]
```

`mydate` provides a "user-friendly" version of the `date` command. The available options are:
`-d` Only print a line concerning date information.
`-t` Only print a line concerning time information.
The default action of `mydate` is to print both lines.
Examples

```
$ mydate
The time is 16:19:42
Today's date is Friday the 16th of April, 1999

Today's date is Friday the 16th of April, 1999

$ mydate -t
The time is 16:19:58

$ mydate -d -t
The time is 16:20:05
Today's date is Friday the 16th of April, 1999
```

13.42 Shell Programming Solutions

1. (a) To print the number of current users:

   ```
   echo There are currently ‘who|wc -l‘ users of the system.
   ```

   (b) The logon details of a user:

   ```
   # list a users activity
   who | grep $1
   ```

   (c) A `.profile` file could look like:

   ```
   # Print greeting
   echo Hello again ‘who am i‘

   # Set up the prompt
   export PS1="\u $ "

   # Output the date
   echo The date is ‘date‘

   # How much mail?
   echo You have ‘grep -c "^From " /var/spool/mail/lee‘ mails.
   ```

2. (a) You should find that commands which *work* have an exit status of 0. Programs which do not work have some non-zero exit status. The grep command uses a common distinction that 1 means a command ran and failed, whereas 2 means it did not succeed in running.
   You should have seen something like this :

   ```
   $ grep root /etc/passwd
   root:x:0:0:root:/root:/bin/bash
   $ echo $?
   0
   $ grep nonuser /etc/passwd
   $ echo $?
   1
   $ grep root /etc/paswd
   grep: /etc/paswd: No such file or directory
   $ echo $?
   2
   ```

   (b) A solution could be :

   ```
   echo There are $# args
   for i in "$@"
   do
     echo $i
   done
   ```

(c) You should see something like this:

```
$ arguments one two 'three four'
There are 3 arguments.
one
two
three four
$ arguments one ' ' three
There are 3 arguments.
one

three
$ arguments a*
There are 1 arguments.
arguments
```

The 'three four' on the first command line is treated as one argument, and not split into two on the space because of the surrounding quotes ('). The null argument ' ' in the second example is an explicit null argument and is thus retained. The third example shows how filename generation is applied by the shell when creating the argument list.

(d)
```
for i in 0 1 2 3 4 5 6 7 8 9
do
  for j in 0 1 2 3 4 5 6 7 8 9
  do
    echo $i$j
  done
done
```

Note that this is a lot more cumbersome than the equivalent loops in most programming languages, but it is not the sort of work that the shell was designed for.

3. (a) The save shell script:

```
if test  ! -d $HOME/backup
then
  if mkdir $HOME/backup
  then
    :
  else
  echo Can't create $HOME/backup >&2
    exit 2
  fi
fi

cp $* $HOME/backup
exit 0
```

(b)
```
#whoare - checks status of users
if test $# -eq 0
then
  echo usage: whois name [ name . . . ]
  exit 2
fi

for user
```

```
do

  if who | grep $user >/dev/null
  then
    echo $user is logged on
  if grep $user /etc/passwd >/dev/null
  then
    echo $user is a user
  else
    echo $user does not exist
  fi
done
```

4. (a) A simple solution to mydate is:

```
# mydate
if [ $# -eq 0 ]
then
                D="YES"
                T="YES"
fi

for i in $*
do
                case $I in
                -t) T="YES" ; ;
                -d) D="YES" ; ;
                *) echo invalid argument ignored ; ;
        esac
done

set 'date'

if [ "$T" = "YES" ]
then echo The time is $4
fi

if [ "$D" = "YES" ]
then echo Todays date is $1 the $3 of $2, $6
fi
```

One full solution to mydate is:

```
# mydate
if [ $# -eq 0 ]
then
                in="-t -d"
else
                in=$*
fi

for i in $in
do

case $i in
        -t) set 'date'
        echo The time is $4; ; ;

        -d) set 'date' + %w %m' '
                case $1 in

0) day=Sunday ; ;
1) day=Monday ; ;
2) day=Tuesday ; ;
3) day=Wednesday ; ;
4) day=Thursday ; ;
5) day=Friday ; ;
6) day=Saturday ; ;
esac
case $2 in

0) month=January ; ;
1) month=February ; ;
2) month=March ; ;
3) month=April ; ;
4) month=May ; ;
5) month=June ; ;
6) month=July ; ;
7) month=August ; ;
8) month=September ; ;
9) month=October ; ;
10) month=November ; ;
11) month=December ; ;
*)      echo 2nd arg is $2; ; ;

        esac
        set 'date'
        case $3 in
1) end=st ; ;
2) end=nd ; ;
3) end=rd ; ;
21) end=st ; ;
22) end=nd ; ;
23) end=rd ; ;
31) end=st ; ;
*)      end=th ; ;
        esac
echo Today\'s date is $day the $3$end of $month, $6; ;;

*)      echo invalid argument ignored; ; ;

        esac
done
```

Module 14

Special Topics

Objectives

After completing this module, you should be able to:

- Understand and configure the LILO boot loader
- Use and verify RPMs
- Build and install applications from sources
- Understand and use key elements of the `/proc` fs

14.1 The Linux Bootloader (LILO)

- The $\boxed{\text{LI}}$nux Boot $\boxed{\text{LO}}$ader (`LILO`) is a versatile beast
- Doesn't depend on any a specific filesystem
- Can boot Linux kernels off floppies, hard disks etc.
- Can act as ‘boot manager’ for other OSs
- Can select from up to 16 images at boot time
- Parameters (e.g. boot device) set independently for each image
- Can replace the Master Boot Record (MBR)

14.2 LILO configuration

- LILO takes command line options
- Configuration lives in `/etc/lilo.conf`
- Confirm changes by running `/sbin/lilo -v`
  - ♢ Machine may be unbootable without this
- An example:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
image=/boot/vmlinuz-2.0.34-0.6
      label=linux
      root=/dev/hdb1
      initrd=/boot/initrd
      read-only
      append="mem=256M"
image=/boot/vmlinuz-2.0.35
      label=mplinux
      root=/dev/hdb1
      initrd=/boot/initrd.old
      read-only
image=/boot/vmlinuz-2.2.7
      label=linux227
      root=/dev/hdb1
      initrd=/boot/initrd.2.2.7
      read-only
other=/dev/hda1
      label=dos
      table=/dev/hda
```

```
        loader=/boot/chain.b
```

14.3 Understanding `lilo.conf`

- Line 1 tells LILO:
  - ♢ which *partition* contains the *boot sector*

    or
  - ♢ which *device* contains the MBR
- Line 2 identifies the *map file*

  i.e. the file which tells LILO where to find all the files needed to boot an OS
- Line 3: The boot loader itself, loads into the BIOS then loads the selected kernel
- Line 4: Ensures a prompt for OS selection, etc
- Line 5: Wait time before booting default kernel
- Line 6: Message file shown before prompt
- Lines 7-22: Detail specific bootable Linux kernels
  - ♢ Line 7: Points to the default Linux kernel
  - ♢ Lines 13,18: Point to alternative Linux kernels
  - ♢ Lines 8,14,19: Commands to start each kernel
  - ♢ Lines 9,15,20: Root partition for each kernel
  - ♢ Lines 10,16,21: Files to load as initial ramdisks
  - ♢ Lines 11,17,22: Mount root partition read-only so `fsck` can run at boot time
- Lines 23-26: Detail a foreign OS
- Line 23: Points to the bootable partition or device
- Line 24: Names command to start foreign OS
- Line 25: Location of foreign OS partition table
- Line 26: Chain loader for booting OS kernel
  - ♢ Defaults to first hard disk partition

14.4 Red Hat Package Management Tool (`rpm`)

- `rpm` is used by most Linux distributions to:
  - ♢ Install, upgrade, and uninstall packages
  - ♢ Query files within packages
  - ♢ Assign and verify package signatures
  - ♢ Maintain installed packages and database
  - ♢ Set permissions on packaged files
  - ♢ Build packages (Special Topics module)
- Can operate via ftp or web
- Supports two kinds of package:
  - ♢ Pre-compiled
  - ♢ Source
- Graphic front-ends, of varying quality, e.g.
  - ♢ `gnorpm`
  - ♢ `kpackage` (also handles `.deb`)
  - ♢ `glint`
- Other distributions will have something similar
  - ♢ Debian has `.deb` files
  - ♢ Stampede has `.slp` packages

14.5 Install, Upgrade and Uninstall with `rpm`

- SYNTAX:

```
$ rpm [options] package-file
```

- *package-file* may be filename or URL
- Common options:

Option	Effect
`-i`	Install package
`-U`	Upgrade package
`-e`	Erase (uninstall) package
`-F`	Freshen (only upgrade if earlier version exists)
`--force`	Same as `--replacepkgs` or `--replacefiles` (install even if packages/files replace already installed ones) or `--oldpackage` (let upgrade replace newer package). Dangerous
`--percent`	Print percentages as files are unpacked; makes `rpm` easier to run from other tools
`--nodeps`	No dependency check. Dangerous
`--test`	Don't install, just check/report potential conflicts
`--relocate` *`oldpath=newpath`*	Relocatable packages to *newpath*. Useful if packager sets innapropriate paths

14.6 Query Options for `rpm`

- SYNTAX:

```
$ rpm -q [query-options] package-file
```

- *package-file* may be filename or URL
- Common options:

Option	Effect
`-a`	Query all installed packages
`-f` *file*	Show package owning *file*
`-p` *package-file*	Show (uninstalled) *package-file*. May be a URL
`--provides`	Show capabilities package provides
`-i`	Show package info, including name, version, description. Uses `--queryformat` if specified
`-R`	Show packages required by this one
`-l`	List files in package
`-s`	Show file state (normal, not installed, replaced)
`--scripts`	Show shell scripts used for un/installation
`--dump`	Dump file info as follows: `path size mtime md5sum mode owner group isconfig isdoc rdev symlink`. Must be with one of `-l, -c, -d`

14.7 Verify Options for `rpm`

- Compares info about installed files with original package info stored in rpm database
- Compares size, MD5 sum, permissions, type, owner, group, etc. Discrepencies are displayed
- Files not originally installed are ignored
- SYNTAX:

```
$ rpm -V [verify-options] package-file
```

- Common options:

Option	Effect
`--nofiles`	Ignore missing files
`--nomd5`	Ignore MD5 checksum errors
`--nofiles`	Ignore missing files

14.8 Output From the `rpm` Verification Option

- Format:
  1. A string of 8 characters
  2. A possible "c" denoting a configuration file
  3. File name
- For example, verifying your `setup` package will usually report errors like these:

```
S.5....T c /etc/exports
S.5....T c /etc/hosts.allow
S.5....T c /etc/hosts.deny
S.5....T c /etc/printcap
S.5....T c /etc/profile
..?..... c /etc/securetty
S.5....T c /etc/services
```

- Each character shows result of a comparison of one file attribute with the value of that attribute recorded in the RPM database
- `.` (full-stop) indicates the test passed
- The following characters denote failure of specified tests:

`5`	MD5 sum
`S`	File size
`L`	Symlink
`T`	Mtime
`D`	Device
`U`	User
`G`	Group
`M`	Mode (includes permissions and file type)
`?`	Couldn’t complete check

14.9 Building And Installing Applications

- Linux/Unix has often been called unfriendly
- Package management techniques and new GUI tools seek to remedy this
- Sometimes necessary to compile packages from source
- Common with security updates
  - ♢ Code fixes available before new packages
  - ♢ Best to update as quickly as possible

14.10 `Autoconf`

- Recent years have made compilation easier
- Tools called `autoconf` and `automake`
- Set up by package author/maintainer
  - ♢ Can determine your system type
  - ♢ Check for presence of needed libraries
  - ♢ Set everything up accordingly
- Provides nicer way to specify compile options
  - ♢ For example, whether to include support for a particular format when building a graphics app

14.11 Using `./configure`

- For a lot of packages the following sequence will install the application

```
$ ./configure
...
$ make
...
$ su
Password:
$ make install
```

- Lots of output is produced giving you valuable information

14.12 Options to configure

- Some ‘standard’ options to `./configure`
- Determine things like:
  - ♢ Place of install
  - ♢ Location of libraries
  - ♢ Turn on/off features
- Examples:

  ```
  $ ./configure --prefix=/home/lee
  ```

  Sets up the build procedure to install the app under `/home/lee` rather than `/usr/local` [1]
- Full list of options

  ```
  $ ./configure --help
  ...
  ```
- Options vary with application
- Commonly see `-with-xxx` or `-enable-xxx` type options

[1] `/usr/local` is the accepted default

14.13 Location of install

- Most applications install under `/usr/local`, e.g.
  - ♢ `/usr/local/bin/gimp`
  - ♢ `/usr/local/man/man1/gimp.1`
- If you're installing libraries you may have to set your `$LD_LIBRARY_PATH` variable
  - ♢ Or run `ldconfig`

  to see the new libraries

14.14 Installation requirements

- Often need to have certain things installed to compile packages
- Most notably the C development libraries
- Also may need other ‘header’ files
  - ♢ `gnome-libs` for compiling Gnome applications
- Can be installed from RPM or sources
- `configure` will check for these
- Tells you if it can’t find what it needs
- Can also differentiate versions of libs/apps
  - ♢ In some cases . . .

14.15 Compilation roundup

- Lack of an RPM isn’t always a problem
- Often just as easy to compile from source
- More power over features
- N.B. RPMs that depend on libraries may not recognize them if they were installed from source!

14.16 The `/proc` filesytem

- Acts as an interface to internal data structures
- Use:
  - ♢ To obtain information about the system
  - ♢ To change certain kernel parameters at runtime
- Also contains one subdirectory for each process running on the system
- Named after the process id (PID) of the process
- Contents of `/proc` can change with different kernel versions
  - ♢ Shouldn't write programs that rely on it

14.17 Process specific subdirectories

- Each process subdirectory has following entries:

`cmdline`	Command line arguments
`cwd`	Link to the current working directory
`environ`	Values of environment variables
`exe`	Link to the executable of this process
`fd`	Directory containing all open file descriptors
`maps`	Memory maps (what memory the process has mapped from files)
`mem`	Memory held by this process
`root`	Link to the root directory of this process
`stat`	Process status
`statm`	Process memory status information
`status`	Process status in human readable form

14.18 Process Status

- To get the status of a process, just read `/proc/PID/status`:

```
$ cat /proc/14502/status
Name:   httpd
State:  S (sleeping)
Pid:    14502
PPid:   381
Uid:    500     500     500     500
Gid:    500     500     500     500
VmSize:     1716 kB
VmLck:         0 kB
VmRSS:       864 kB
VmData:      304 kB
VmStk:        32 kB
VmExe:       300 kB
VmLib:       864 kB
SigPnd: 00000000
SigBlk: 00000000
SigIgn: 00000000
SigCgt: 0040766b
```

- Shows almost the same information as `ps` because `ps` gets its info from `proc`

14.19 Process Memory Usage (`statm`)

- The `statm` file details process memory usage
- Its values have the following meanings:

`size`	total program size
`resident`	size of in memory portions
`shared`	number of the pages that are shared
`trs`	number of pages that are ’code’
`drs`	number of pages of data/stack
`lrs`	number of pages of library
`dt`	number of dirty pages

- Ratio `trs/data/library` is only approximate

14.20 Kernel data

- The following subdirectories give info on the running kernel
- Not all present on every system
  - ♢ depends on kernel config and loaded modules

`apm`	Advanced power management info
`cmdline`	Kernel command line
`cpuinfo`	Info about the CPU
`devices`	Available devices (block and character)
`dma`	Used DMA channels
`filesystems`	Supported filesystems
`interrupts`	Interrupt usage
`ioports`	I/O port usage
`kcore`	Kernel core image
`kmsg`	Kernel messages
`ksyms`	Kernel symbol table
`loadavg`	Load average
`locks`	Kernel locks
`meminfo`	Memory info
`misc`	Miscellaneous
`modules`	List of loaded modules
`mounts`	Mounted filesystems
`partitions`	Table of partitions known to the system
`rtc`	Real time clock
`slabinfo`	Slab pool info
`stat`	Overall statistics
`swaps`	Swap space utilization
`uptime`	System uptime
`version`	Kernel version

14.21 Interrupts In Use

- See `/proc/interrupts` to:
  - ♢ Check which interrupts are currently in use
  - ♢ Check what they are used for/by
- For example:

```
$ cat /proc/interrupts
 0:  310441744   timer
 1:        442   keyboard
 2:          0   cascade
 3:     259823 + serial
 4:       1974 + serial
 5:    5618945 + serial
 8:          1 + rtc
10:  140843983   3c509
13:          1   math error
14:  272072142 + ide0
15:  307923561 + ide1
```

14.22 IDE Devices (`/proc/ide`)

- Details all IDE devices known to the kernel
- One subdirectory for each device
- Each directory containing these files:

`cache`	The cache
`capacity`	Capacity of the medium
`driver`	Driver and version
`geometry`	Physical and logical geometry
`identify`	Device identify block
`media`	Media type
`model`	Device identifier
`settings`	Device setup
`smart_thresholds`	IDE disk management thresholds
`smart_values`	IDE disk management values

14.23 Networking (`/proc/net`)

- The files and their meanings:

`arp`	Kernel ARP table
`dev`	Network devices with statistics
`dev_mcast`	Lists the Layer2 multicast groups a device is listening to (interface index, label, number of references, number of bound addresses).
`dev_stat`	Network device status
`ip_fwchains`	Firewall chain linkage
`ip_fwnames`	Firewall chains
`ip_masq`	Directory containing the masquerading tables
`ip_masquerade`	Major masquerading table
`netstat`	Network statistics
`raw`	Raw device statistics
`route`	Kernel routing table
`rpc`	Directory containing rpc info
`rt_cache`	Routing cache
`snmp`	SNMP data
`sockstat`	Socket statistics
`tcp`	TCP sockets
`tr_rif`	Token ring RIF routing table
`udp`	UDP sockets
`unix`	UNIX domain sockets
`wireless`	Wireless interface data (Wavelan etc)
`igmp`	IP multicast addresses, which this host joined
`psched`	Global packet scheduler parameters
`netlink`	List of PF_NETLINK sockets
`ip_mr_vifs`	List of multicast virtual interfaces
`ip_mr_cache`	List of multicast routing cache
`udp6`	UDP sockets (IPv6)
`tcp6`	TCP sockets (IPv6)
`raw6`	Raw device statistics (IPv6)
`igmp6`	IP multicast addresses, which this host joineed (IPv6)
`if_inet6`	List of IPv6 interface addresses
`ipv6_route`	Kernel routing table for IPv6
`rt6_stats`	global IPv6 routing tables statistics
`sockstat6`	Socket statistics (IPv6)
`snmp6`	Snmp data (IPv6)

14.24 Networking 2 (`/proc/net`)

- Use `/proc/net` to see:
  - ♢ The network devices available in your system
  - ♢ How much traffic is routed over them

- For example:

```
$  cat /proc/net/dev
Inter-|   Receive                   |  Transmit
 face |packets errs drop fifo frame|packets errs drop fifo colls carrier
    lo:20664014    0    0    0    0 20664014    0    0    0     0    0
  eth0:93964796 618018 618018 616771 618018 83123181    1    0    0 49304   39
eth0:0:    141    0    0    0    0        1    0    0    0     0    0
eth0:1:    333    0    0    0    0        4    0    0    0     0    0
eth0:2:      0    0    0    0    0        0    0    0    0     0    0
eth0:3:      0    0    0    0    0        0    0    0    0     0    0
eth0:4:    212    0    0    0    0        3    0    0    0     0    0
```

14.25 SCSI info (/proc/scsi)

- To see a list of all recognized SCSI devices in `/proc/scsi`:

```
$ cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: QUANTUM  Model: XP34550W         Rev: LXY4
  Type:   Direct-Access                    ANSI SCSI revision: 02
Host: scsi0 Channel: 00 Id: 01 Lun: 00
  Vendor: SEAGATE  Model: ST34501W         Rev: 0018
  Type:   Direct-Access                    ANSI SCSI revision: 02
Host: scsi0 Channel: 00 Id: 02 Lun: 00
  Vendor: SEAGATE  Model: ST34501W         Rev: 0017
  Type:   Direct-Access                    ANSI SCSI revision: 02
Host: scsi0 Channel: 00 Id: 04 Lun: 00
  Vendor: ARCHIVE  Model: Python 04106-XXX Rev: 703b
  Type:   Sequential-Access                ANSI SCSI revision: 02
```

- One file for each adapter found in the system
- Info on controller, IRQ used, IO address range:

```
$ cat /proc/scsi/ncr53c8xx/0
General information:
 Chip NCR53C875, device id 0xf, revision id 0x4
 IO port address 0xec00, IRQ number 11
 Synchronous period factor 12, max commands per lun 4
```

14.26 Parallel Port (`/proc/parport`)

- Info on parallel ports
- One subdirectory for each port
- named after the port number (0,1,2, . . .)
- Contains four files:

`autoprobe`	Autoprobe results of this port
`devices`	Connected device modules
`hardware`	Port type, io-port, DMA, IRQ, etc
`irq`	Used interrupt, if any

14.27 Kernel Parameters (`/proc/sys`)

- Displays parameters within the kernel
- Allows you to *change* them
- Can tune and monitor kernel operation
- Be very careful, a reboot may be the only option after a mistake
- To change a value `echo` the new value into the file (see file handles example below)
- Superuser permission is required
- Can be automated via the init scripts
  - ◊ Should check kernel documentation when upgrading kernel to check the `/proc` information you use has not changed

14.28 File system data (`/proc/fs`)

- Info on file handles, inodes, dentry and quotas
- `/proc/sys/fs` currently contains these files:

`dentry-state`	Status of the directory cache
`dquot-nr`	Number of allocated and free disk quota entries
`dquot-max`	Maximum number of cached disk quota entries
`file-nr`	Number of allocated, used and maximum number of file handles
`file-max`	Maximum number of file handles that the Linux kernel will allocate
`inode-state`	Contains three actual numbers and four dummy values. Actual numbers are `nr_inodes` (inodes allocated), `nr_free_inodes` (free inodes), and `preshrink` (nonzero when the `nr_inodes` > `inode-max` and system needs to reduce inode list instead of allocating more)
`inode-nr`	Contains the first two items from `inode-state`
`inode-max`	Maximum number of inode handlers. Should be 3-4x > `file-max`, since stdin, stdout, and network sockets also need an `inode struct` to handle them
`super-nr`	Number of currently allocated super block handlers
`super-max`	Maximum number of super block handlers. Every mounted file system needs one, so more mounts need more of them

14.29 Example: Increase Maximum Filehandles

- Kernel allocates file handles dynamically, but doesn’t free them while processes still run
- The default value maximum (`file-max`) is 4096
- To change it, just write a new number into the file:

```
# cat /proc/sys/fs/file-max
4096
# echo 8192 > /proc/sys/fs/file-max
# cat /proc/sys/fs/file-max
8192
```

- Useful for all customizable kernel parameters
- N.B. There is still a per process limit of open files (1024 by default) — can’t be easily changed [2]

[2] To change it, edit the files `limits.h` and `fs.h` in the directory `/usr/src/linux/include/linux`. Change the definition of NR_OPEN and recompile the kernel.

14.30 General Kernel Parameters (`/proc/sys/kernel`)

- There are many general prarameters here and they vary from system to system
- The most commonly utilised covers the behaviour of `ctrl-alt-del`
  - ◇ When = 0, `ctrl-alt-del` is trapped and sent to `init(1)` to handle a graceful restart
  - ◇ When > 0, Linux produces an immediate reboot, without syncing dirty buffers
  - ◇ Occasionally `ctrl-alt-del` won't reach the kernel (e.g. intercepted by `dosemu`)
- Other files you might see, include:
  - ◇ `acct`
  - ◇ `domainname and hostname`
  - ◇ `osrelease, ostype and version`
  - ◇ `panic`
  - ◇ `sg-big-buff`
  - ◇ `modprobe`

14.31 Virtual Memory Subsystem (`/proc/sys/vm`)

- Typically used to set rather than read parameters
- Used for low-level tuning of the kernel’s virtual memory (VM) subsystem
- Generally for wizards, i.e. supra-guru

14.32 Device Specific Parameters (`/proc/sys/dev`)

- A newish feature
- May not even exist on some systems
- Currently only support for CDROM drives
- Only one read-only file on CD-ROM drives attached to the system, e.g.

```
$ cat /proc/sys/dev/cdrom/info
CD-ROM information

drive name:           sr0  hdc
drive speed:           0    6
drive # of slots:      1    0
Can close tray:        1    1
Can open tray:         1    1
Can lock tray:         1    1
Can change speed:      1    1
Can select disk:       0    1
Can read multisession: 1    1
Can read MCN:          1    1
Reports media changed: 1    1
Can play audio:        1    1
```

- Example shows two drives, `sr0` and `hdc` with their features

14.33 Remote Procedure Calls (`/proc/sys/sunrpc`)

- Contains four files, enabling or disabling debugging for the RPC functions:
  - ♢ NFS
  - ♢ NFS-daemon
  - ♢ RPC
  - ♢ NLM
- Default values are 0
- Can be set to 1 to turn debugging on

14.34 Networking (`/proc/sys/net`)

- The interface to the networking parts of the kernel is located in `/proc/sys/net`
- Contains literally hundreds of parameters which can be read or set
- This table shows all possible subdirectories, some will not appear on every system:

core	General parameter	appletalk	Appletalk protocol
unix	Unix domain sockets	netrom	NET/ROM
802	E802 protocol	ax25	AX25
ethernet	Ethernet protocol	rose	X.25 PLP layer
ipv4	IP version 4	x25	X.25 protocol
ipx	IPX	token-ring	IBM token ring
bridge	Bridging	decnet	DEC net
ipv6	IP version 6		

- No time to discuss them all here

14.35 IPV4 settings (`/proc/sys/net/ipv4`)

- ICMP settings:
  - ◇ `icmp_echo_ignore_all` and `icmp_echo_ignore_broadcasts`

    Turn on (1) or off (0). First ignores `ping` of your host. Second ignores `pings` of your network. Can help tackle denial of service packet flooding attacks

  - ◇ `icmp_destunreach_rate` `icmp_echoreply_rate` `icmp_paramprob_rate` `icmp_timeexeed_rate`

    Set limits for sending ICMP packets to specific targets, depending on icmp type, i.e. can stop packet flooding *from* your host
- There are dozens of other IP and TCP settings . . . too many to discuss here
- See `/usr/src/linux/Documentation/proc.txt` for details

14.36 Special Topics Exercises

1. *Configuring LILO*
   (a) Put a copy of your existing Linux kernel on a floppy, then configure `lilo` to boot your machine from it. N.B. Do NOT do the next question until you are sure your boot disk works!
   (b) Configure `lilo` to boot your machine from a new Linux kernel on your hard drive.
   Ideally you should do this with a distinctively new kernel, such as the one made for the Kernel Internals module, but you could simply copy your current kernel with a new name.
2. *Using RPMs*
   (a) Use `rpm` from the command line to:
      i. Install a package
      ii. Update a package
      iii. Uninstall a package
   (b) If you have a distribution CD available:
      i. Find the main directory containing RPMs.
      ii. Work out and use the command string to put a complete list of all the packages' summary information and filenames into a file called `rpmlist.txt`
   (c) Verify your `setup` RPM.
   (d) With a colleague, draw up a list of other RPM packages containing files which have probably changed since installation. Verify them.
   (e) Imagine you suspect a system break-in has occurred. Use `rpm` to check:
      i. Whether such a break-in has occurred
      ii. How your files have been affected
   (f) Depending on what you have on your system, find out which packages are required to run `fvwm2` or another window manager
3. *Building And Installing Applications From Sources*
   (a) Install an application from sources provided, or indicated, by your tutor
4. *Using the `/proc` filesystem*
   (a) Print (to screen) simple info from `/proc` on:
      i. memory usage
      ii. cpu usage
   (b) Use `/proc` to get status info on the following processes:
      i. The shell you are currently working in
      ii. `syslogd`
      iii. `crond`
   (c) Use `/proc` to enable/disable:
      i. IP forwarding

    ii. ICMP packet flooding from your host
    iii. ICMP packet flooding of your network

(d) Pass parameters to the running kernel to:

    i. Increase the maximum number of file handles available
    ii. Change your hostname

    N.B. Change back to your original hostname as soon as you have succeeded. Many other exercises on your course may depend on it.

14.37 Special Topics Solutions

1. *Configuring LILO*

   (a) Put a boot image on the floppy, then add something like the following to `lilo.conf`, before running `lilo` and rebooting:

   ```
   image=/boot/bzlinuz
           label=floppylinux
           root=/dev/fd0
           read-only
   ```

   (b) Put a boot image in the `boot` directory of your hard disk, then add something like the following to `lilo.conf`, before running `lilo` and rebooting:

   ```
   image=/boot/newlinuz
           label=newlinux
           root=/dev/hda1
           read-only
   ```

2. *Using RPMs*

   (a) Use something like the following commands:

      i. `$ rpm -i` *`package`*

      ii. `$ rpm -U` *`package`*

      iii. `$ rpm -e` *`package`*

   (b) If you have a distribution CD available:

      i. On Red Hat distributions it will usually be `/mnt/cdrom/RedHat/RPMS/`

      ii. `$ rpm -qilp *.rpm > rpmlist.txt`

   (c) `$ rpm -V setup`

   (d) Potentially hundreds of correct answers to this one. Dependent on host setup. On any system, the following files should really have changed:

      - `passwd`
      - `group`
      - `hosts.allow`
      - `hosts.deny`

      Find out which package these belong to using:

      `$ rpm -qf` *`filename`*

   (e) `$ rpm -Va`

   (f) `$ rpm -R` *`package`*

3. *Building And Installing Applications From Sources*

   There are several possible methods, but the most popular procedure does the following in the source directory:

   ```
   $ ./configure
   ...
   $ make
   ...
   $ su
   Password:
   $ make install
   ```

4. *Using the `/proc` filesystem*

   (a) i. `$ cat /proc/meminfo`

       ii. `$ cat /proc/cpuinfo`

   (b) Use `ps` or `top` to get the appropriate process IDs, then:

   ```
   $ cat /proc/PID/status
   ```

   (c) i.
       - On: `$ echo 1 > /proc/sys/net/ipv4/ip_forward`
       - Off `echo 0 > /proc/sys/net/ipv4/ip_forward`

       ii. See tutor

       iii. See tutor

   (d) E.g.

       i. `$ echo 8192 > /proc/sys/fs/file-max`

       ii.
       - Change: `$ echo newname > /proc/sys/kernel/hostname`
       - Undo: `$ echo orignalname > /proc/sys/kernel/hostname`

Module 15

Fundamentals of TCP/IP

Objectives

This module is intended as an introduction to the the basic concepts of IP networking. By the end of it you should understand:

- The history and uses of various protocols
- How subnetting and netmasks work
- About interfaces
- The use of ports

15.1 Fundamentals of TCP/IP Networking

Key concepts:

- Packets
- TCP vs UDP
- Services
- Subnetting inc /xx form
- Routing

15.2 History

- Developed by ARPA for university & military research
- Robust, reliable, wide area network protocol, system-independent
- Will route traffic around network outages (if routing protocols used)
- Came into widespread use in mid-late 1970s
- Popularity hugely helped by free availability of the BSD Unix implementation

  i.e. the pre-Linux reference platform
- Now the standard protocol - the Internet based totally upon it

15.3 Recap of basic IP Concepts - Components

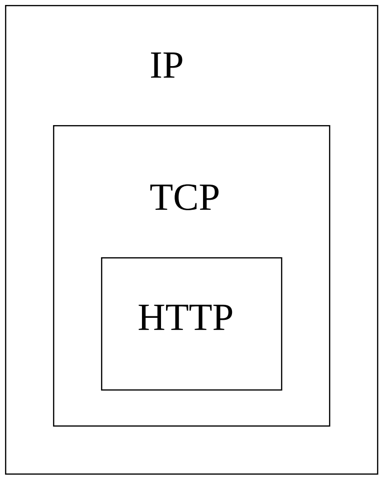
- Properly, The Internet Protocol Suite (IP Suite)
- Usually erroneously referred to as TCP/IP
- Consists of numerous protocols
- IP is used to encapsulate:
  - ♢ TCP (Transmission Control Protocol)
  - ♢ UDP (User Datagram Protocol)
  - ♢ ICMP (Internet Control Message Protocol)
  - ♢ other routing & management protocols

15.4 IP versions

- Currently at Version 4 (IPV4)
  - ♢ Entire Internet based on IPV4
  - ♢ Quickly running out of spare numbers
- IPV6 well standardised
  - ♢ Important improvements
  - ♢ Currently in miniscule use
  - ♢ Migration will occur eventually
  - ♢ Support already in Linux

15.5 Packets

- All data transferred in packets (datagrams)
- Each packet contains various flags & admin information
  - ♢ Source address (32 bits)
  - ♢ Destination address (32 bits)
- Addresses identify hosts
  - ♢ Usually an interface on a host
- Addresses are the basis of packet routing
- Packets can be split reassembled, differentially routed, arrive out-of-order or just get lost
- Higher-level protocols (e.g. TCP) add sequencing reliability, flow control etc.

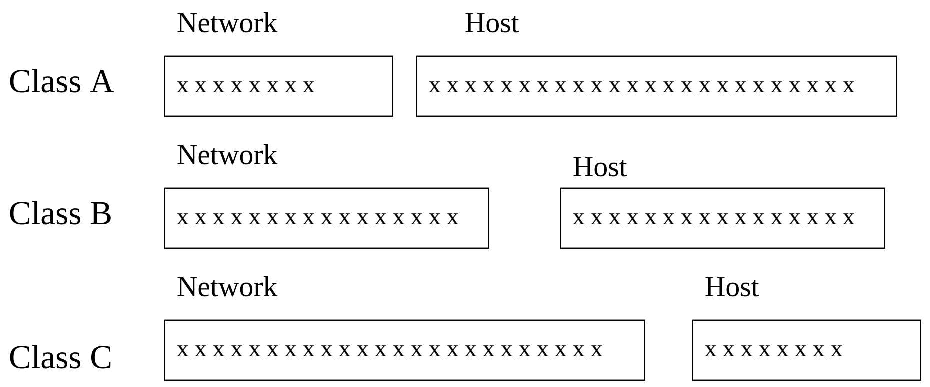


15.6 Addresses

- Addresses shown in ‘dotted decimal’ - break into 4 bytes
  - ◇ 192.168.0.129
- Four address families
  - ◇ Class A 0.x.x.x-127.x.x.x
  - ◇ Class B 128.x.x.x-191.x.x.x
  - ◇ Class C 192.x.x.x-223.x.x.x
  - ◇ ’reserved’ 224.x.x.x
- Class A network 127 is special
  - ◇ Refers to the current network (any network)
  - ◇ Current host is *always* 127.0.0.1
  - ◇ *‘loopback’* address

15.7 Addresses (continued)

- Addresses identify:
  - ◇ Network (used for routing between networks)
  - ◇ Hosts on a particular network
    - ∗ Class A 8 network bits, 24 host bits
    - ∗ Class B 16 network bits, 16 host bits
    - ∗ Class C 24 network bits, 8 host bits
- In all networks, host-parts of all zeros (0) and all ones (255) are reserved
  - ◇ Host-part zero refers to the network itself
  - ◇ Host-part all ones is 'broadcast' address (all hosts)



15.8 Netmasks and subnetting

- Netmasks split host and network part of address
- Says which machines can be reached directly
- Example:

```
Netmask 255     .255     .255     .0
Binary  11111111.11111111.11111111.00000000

IP      192     .168     .0       .129
Binary  11000000.10101000.00000000.10000001
```

- To work out the network part

```
Netmask  11111111.11111111.11111111.00000000
IP       11000000.10101000.00000000.10000001
--------------------------------------------
Result   11000000.10101000.00000000.00000000
--------------------------------------------
            192       168      0        0
```

- To work out the host part

```
Netmask  11111111.11111111.11111111.00000000
IP       11000000.10101000.00000000.10000001
--------------------------------------------
Result   00000000.00000000.00000000.10000001
--------------------------------------------
             0         0       0       129
```

15.9 Netmasks with `/xx`

- Sometimes see IP addresses given as :

  `192.168.0.129/24`

- `/xx` is another form of netmask. Says that the left-most `xx` bits specify the network
  - ♢ i.e. /24 means 24 ‘1s’, a netmask of:

    `11111111.11111111.11111111.00000000`

    or

    `255.255.255.0`

- Was rare, but becoming more common
- Examples:
  - ♢ 10.0.0.0 is network 10
  - ♢ 192.168.5.0 is network 192.168.5
  - ♢ 10.255.255.255 is broadcast on network 10
  - ♢ 192.168.5.255 is broadcast on network 192.168.5

15.10 Transferring Data

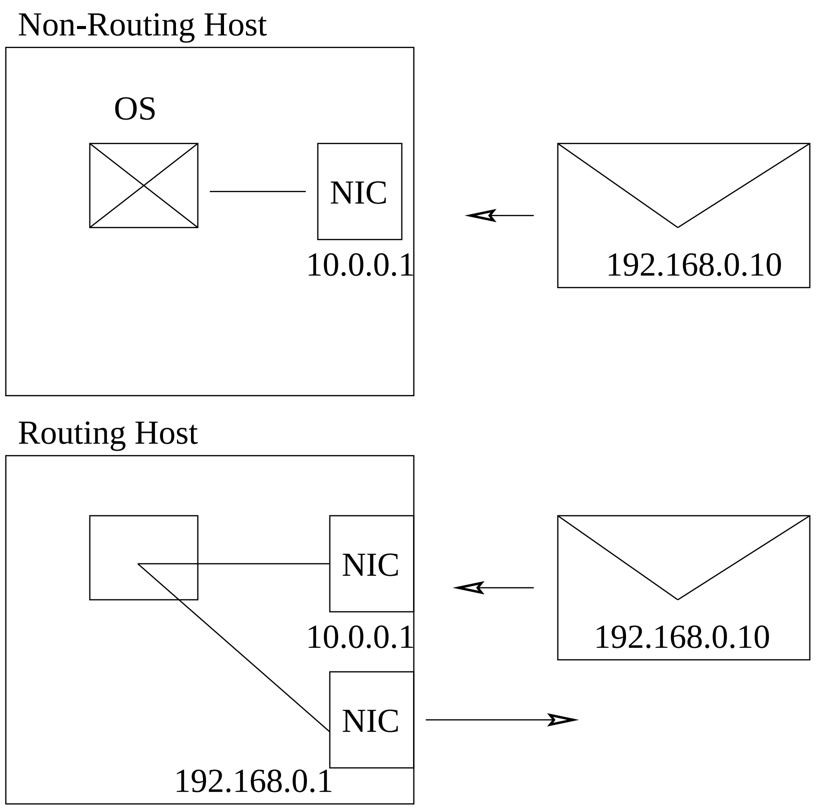
- IP allows datagrams to be sent and routed between hosts
- Contains no *application-level* data
- Data part will be one of UDP, TCP, ICMP etc.
- TCP is ‘session’ oriented data, used for long-lived connections
- UDP used for fire-and-forget messages
- ICMP used for control & testing, not seen by most applications or users
- Examples:
  - ♢ Email transferred using SMTP over TCP, (maybe many bytes, order important)
  - ♢ Web pages use HTTP over TCP
  - ♢ UDP more obscure, used for `NFS`
  - ♢ ICMP: ‘`ping`’ utility, used to test visibility

15.11 Hosts & Interfaces

- Hosts are individual computers/systems
- Each host has one or more interfaces
  - ♢ Each interface is a point of connection to a network (often a NIC or modem)
- Many hosts have a single interface, so the address is the host
- May have more than one interface
  - ♢ Interfaces could be on different networks
  - ♢ Can act as routers, forwarding packets
- Each ‘interface’ will have a single address

15.12 Routing

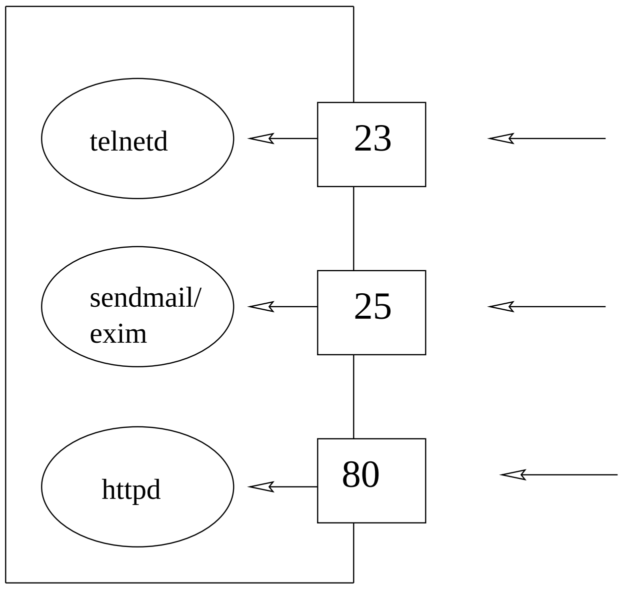
- Hosts receive packets on one or more interfaces
- Check to see if packet is for current host
  - ♢ If so, deliver to the UDP/TCP etc mechanisms
- Otherwise
  - ♢ If routing enabled [1]
    - ∗ Forward packet to appropriate host
  - ♢ Routing based on internal routing table
  - ♢ Manipulated by `route` command
    - ∗ Superuser only



[1]Often referred to as 'IP forwarding'

15.13 Ports

- Not enough just to deliver packets to hosts
- Deliver to correct applications on the host
  - ♢ Hosts presumed to be multitasking
- UDP & TCP both include port numbers
  - ♢ 16 bit numbers (0-65535)
  - ♢ Each UDP/TCP packet contains source & destination port
  - ♢ sourceport/sourceaddress & destinationport/destinationaddress uniquely identify a conversation



15.14 Ports cont..

- Many ‘well known’ ports published for client-server applications
- See `/etc/services` under Linux
  - ♢ TCP/25 - SMTP mail
  - ♢ TCP/23 - telnet (remote terminal access)
  - ♢ TCP/80 - HTTP (web protocol)
- Unix-like systems reserve ports below 1024 for super-user
- Ordinary users cannot run ‘special’ services without authorisation
- This cannot be trusted in other environments, such as Windows

15.15 Exercises

1. Using `ifconfig`, explore the interfaces available on your current system
2. Discover the IP addresses of some other machines on your network and check that you can ping them all. What `class` (A, B or C) of network are they on?
3. From the man page for `ping`, discover how to set a regular ping running every five seconds. Then investigate how you can send extra-long ping packets (try sending a ping longer than 2K bytes).
4. What ports and protocols are used to run the following services?
   - Telnet
   - SMTP
   - Printer
   - Talk
5. What happens if you `telnet` to various ports? (Try 25 or 110)
6. Use this fact to discover what mail system your machine runs, and see if it runs a webserver (Port 80)

15.16 Solutions

1. `ifconfig` by default shows a list of the currently configured interfaces including the IP addresses and netmasks.

2. `ping 10.0.0.2` will send pings to the `10.0.0.2` interface provided routing is set up correctly. You should be able to find out what class of network you are on from the IP address. See section 15.6 for details.

3. To send a regular ping every 5 seconds use

   ```
   $ ping -i 5
   ```

   To alter the packet size you use the `-s option` to give a size in bytes

   ```
   $ ping -s 3000
   ```

4. The following values are taken from `/etc/services`

Port	Service
23	Telnet
25	SMTP
515	Printer
517	Talk

5. You should be able to talk directly to the daemon at the other end, e.g.

   ```
   $ telnet localhost 110
   Trying 127.0.0.1...
   Connected to localhost.
   Escape character is '^]'.
   +OK POP3 localhost v4.47 server ready
   USER lee
   +OK User name accepted, password please
   PASS ********
   +OK Mailbox open, 2 messages
   RETR 1
   +OK 332 octets
   Return-Path: <lee>
   Received: (from lee@localhost)
           by gbdirect.co.uk (8.8.7/8.8.7) id LAA12997
           for lee@localhost; Mon, 14 Feb 2000 11:39:19 GMT
   Date: Mon, 14 Feb 2000 11:39:19 GMT
   From: Lee <lee@gbdirect.co.uk>
   Message-Id: <200002141139.LAA12997@gbdirect.co.uk>
   To: lee@gbdirect.co.uk
   Subject: Test
   Status: RO

   This is a test.
   .
   QUIT
   +OK Sayonara
   Connection closed by foreign host.
   ```

6. You can sometimes find out what webserver a site is using by telnetting to port 80 and requesting the headers of the main page, e.g.

```
$ telnet www.bbc.co.uk 80
Trying 212.58.224.31...
Connected to www.bbc.net.uk.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 302 Moved Temporarily
Date: Mon, 14 Feb 2000 11:43:06 GMT
Server: Apache/1.3.1 (Unix)
Location: http://www.bbc.co.uk/home/today/
Connection: close
Content-Type: text/html

Connection closed by foreign host.
```

Module 16

Practical TCP/IP

Objectives

After completing this module you should be able to understand and utilise:

- Firewalling principles
- Basic firewalling with `ipchains`
- Network/routing debugging procedures
- Interface configuration under Linux
- The secure shell (`sshd`, `ssh`, and `scp`)

16.1 Ping Protocols

- `ping` used to test network/host availability
- A little about its implementation
  - ♢ Uses ICMP protocol
  - ♢ Send requests of type *echo-request*
  - ♢ Receives answer *echo-reply*

16.2 Network Statistics (`netstat`) in Practice

- Show network status; many options
- Most useful: `-r` and `-n` flags (show routes, numeric addresses only)

```
$ netstat -rn
Kernel IP routing table
Destination   Gateway       Genmask         Flags  MSS Window irtt Iface
192.168.0.0   0.0.0.0       255.255.255.0   U     1500 0         0 eth0
202.80.80.0   192.168.0.1   255.255.255.0   UG    1500 0         0 eth0
192.100.100.0 192.168.0.1   255.255.255.0   UG    1500 0         0 eth0
194.217.100.0 192.168.0.143 255.255.255.0   UG    1500 0         0 eth0
192.168.1.0   192.168.0.1   255.255.255.0   UG    1500 0         0 eth0
192.168.3.0   192.168.0.1   255.255.255.0   UG    1500 0         0 eth0
127.0.0.0     0.0.0.0       255.0.0.0       U     3584 0         0 lo
```

- Note ‘gateway‘ above
  - ♢ route to networks 202.80.80.0, 192.100.100.0, 192.168.1.0 and 192.168.3.0 use gateway 192.168.0.1
  - ♢ 192.168.0.1 is a gateway (router) which knows how to access those networks
  - ♢ route to network 194.217.100.0 is via gateway at 192.168.0.143
- Often see destination of 0.0.0.0
  - ♢ ‘default‘ route
  - ♢ send all otherwise unrouteable packets to designated gateway
- *Iface* column shows which interface will be used
- Note interface for 127.0.0.0 - ‘loopback‘ interface; the host itself

16.3 `netstat` (continued)

- Also show information about connected sockets
- `netstat -a` shows no. of active connections (useful for seeing system load)

-
  ```
  Active Internet connections (servers and established)
  Proto Recv-Q Send-Q Local Address        Foreign Address    State
  tcp        0      0 *:6000               *:*                LISTEN
  tcp        0      0 linux.gazcl:domain   *:*                LISTEN
  tcp        0      0 localhost:domain     *:*                LISTEN
  tcp        0      0 *:linuxconf          *:*                LISTEN
  tcp        0      0 *:auth               *:*                LISTEN
  tcp        0      0 *:finger             *:*                LISTEN
  raw        0      0 *:icmp               *:*                7
  raw        0      0 *:tcp                *:*                7
  Active UNIX domain sockets (servers and established)
  Proto RefCnt Flags       Type       State         I-Node Path
  unix  1      [ ]         STREAM     CONNECTED     8937   @00000082
  unix  1      [ ]         STREAM     CONNECTED     8906   @0000007b
  unix  1      [ ]         STREAM     CONNECTED     8933   @00000081
  unix  0      [ ACC ]     STREAM     LISTENING     327    /dev/log
  unix  1      [ ]         STREAM     CONNECTED     8949   @00000086
  unix  1      [ ]         STREAM     CONNECTED     8926   @0000007f
  unix  1      [ ]         STREAM     CONNECTED     8644   @00000059
  ```

16.4 `netstat` - Further Examples

- Configured interfaces
- `netstat -i`

```
Kernel Interface table
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0 1500   0     0      0      0      0  3107      0      0      0 BRU
lo   3924   0  1035      0      0      0  1035      0      0      0 LRU
```

- `netstat -p` shows processes listening on each socket [1]
  - ♢ Includes PID
  - ♢ Useful to kill processes hogging key ports

```
$ netstat -pn
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address  Foreign Address  State       PID/name
tcp        0      0 10.0.0.1:1025  10.0.0.2:telnet  ESTABLISHED 443/telnet
tcp        0      0 10.0.0.1:1024  10.0.0.3:telnet  ESTABLISHED 442/telnet
tcp        0      0 10.0.0.1:1023  10.0.0.4:ssh     ESTABLISHED 432/ssh
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags  Type    State      I-Node PID/Program name  Path
unix  1      [ ]    STREAM  CONNECTED  662    432/ssh           @00000037
unix  1      [ ]    STREAM  CONNECTED  591    388/login -- lee  @0000002f
```

[1] Only supported in more recent versions

16.5 Network Traffic (`tcpdump`) in Practice

- Used to monitor network traffic
  - ♢ Need sufficient privilege to monitor devices
- Can show only particular information
  - ♢ Traffic to/from a particular host
  - ♢ Traffic on a certain port
  - ♢ Certain types of traffic, e.g. TCP, ARP, UDP
- Very configurable
  - ♢ Decide what you want to do
  - ♢ Then look at manual page

16.6 `tcpdump` Options

- Some options

`-i`	Says which network *interface* to show
`-n`	Print IP addresses *not names*
`-N`	Don't print domain name of address
`-t`	Don't print *timestamp*
`-q`	Show only minimal output (*quiet*)
`-v`	*Verbose* info (time-to-live etc.)

16.7 `tcpdump` Examples

```
$ tcpdump dst host 192.168.0.143 -i eth0 -n -t
tcpdump: listening on eth0
192.168.0.131 > 192.168.0.143: icmp: echo request
192.168.0.131 > 192.168.0.143: icmp: echo request
arp who-has 192.168.0.143 tell 192.168.0.131
192.168.0.131 > 192.168.0.143: icmp: echo request
192.168.0.131.1026 > 192.168.0.143.telnet: S 73945916:73945916(0) win 32120
 <mss 1460,sackOK,timestamp 238586[|tcp]> (DF)
192.168.0.131.1026 > 192.168.0.143.telnet: . ack 3134108710 win 32120 (DF)
192.168.0.131.1026 > 192.168.0.143.telnet: P 0:27(27) ack 1 win 32120 (DF)
192.168.0.131.1026 > 192.168.0.143.telnet: . ack 13 win 32120 (DF)
192.168.0.131.1026 > 192.168.0.143.telnet: P 27:35(108) ack 5 win 32120 (DF)
192.168.0.131.1026 > 192.168.0.143.telnet: P 35:38(3) ack 55 win 32120 (DF)
192.168.0.131.1026 > 192.168.0.143.telnet: P 38:41(3) ack 125 win 32120 (DF)
192.168.0.131.1026 > 192.168.0.143.telnet: . ack 132 win 32120 (DF)
192.168.0.131.1026 > 192.168.0.143.telnet: P 41:42(1) ack 132 win 32120 (DF)
```

```
$ tcpdump dst host 192.168.0.143 -i eth0 -N -q
tcpdump: listening on eth0
09:56:32.947997 landlord.mysql > samosa.5660: tcp 166 (DF)
09:56:32.955822 landlord.mysql > samosa.5660: tcp 166 (DF)
09:56:32.963597 landlord.mysql > samosa.5660: tcp 182 (DF)
09:56:32.970917 landlord.mysql > samosa.5660: tcp 166 (DF)
09:56:32.979341 landlord.mysql > samosa.5660: tcp 166 (DF)
09:56:32.987218 landlord.mysql > samosa.5660: tcp 166 (DF)
09:56:32.995902 landlord.mysql > samosa.5660: tcp 555 (DF)
```

16.8 Firewalling

- Allows you to protect your machine
  - ◊ As well as machines *behind* them
- Checks packet headers before acting on them
  - ◊ Can ignore, reject or accept packets
  - ◊ Makes decision based on source, destination, or packet type
    - ∗ Or a combination
- Set up using `ipchains` under kernel 2.2
  - ◊ Older kernels used `ipfwadm`

16.9 Basic Theory

- Two main considerations
  - ♢ Port Filtering
  - ♢ Host Filtering
- Block services you don’t need
- Limit services you *do* need to specific machines/networks

16.10 Basic Theory (continued)

- Firewalling can be done with `inetd`
  - ♢ `/etc/hosts.allow`
  - ♢ `/etc/hosts.deny`
  - ♢ `/etc/inetd.conf`
- Flaw in `inetd` would still let things through
- Best to drop the packets as soon as possible
  - ♢ Kernel-level filtering

16.11 `ipchains`

- Packet filtering set up using `ipchains`
- All the filtering is done in the kernel
  - ♢ Not by `ipchains`
  - ♢ `ipchains` just sets up/modifies the rules
- All packets entering and leaving are examined [2]

[2] Including loopback traffic which conceptually leaves the machine

16.12 `ipchains` Details

- Every packet goes through one or more '*chains*'
  - ◇ A 'chain' is a set of rules
  - ◇ Rules can accept, reject, or deny a packet
    - ∗ Can also send it to another chain
- Three default chains, *input*, *output*, *forward*
  - ◇ If a packet passes through a default chain without matching:
    - ∗ Fate is determined by the chains *policy*
    - ∗ Can be *Accept*, *deny*, or *reject*
  - ◇ If it reaches the end of a user defined chain
    - ∗ Carries on where it left off
- *forward* is for IP masquerading systems
  - ◇ Not covered here

16.13 `ipchains` Options

- Dealing with chains :

`-N`	Create a new chain
`-X`	Delete an empty chain
`-P`	Change the policy for a chain
`-L`	List the rules in a chain
`-F`	Flush (delete) all rules from a chain

- Dealing with rules :

`-A`	Append a rule to a chain
`-D`	Delete a single rule from a chain
`-I`	Insert a rule at some point in a chain

16.14 Options For Rules

- Use the following to specify packets to match

`-s`	Source address
`-d`	Destination address
`-p`	Protocol (`TCP`, `UDP`, `ICMP`)
`-j` *chain*	Jump to chain/action
`--sport`	Source Port
`--dport`	Destination Port

16.15 `ipchains` - Examples

- In most cases default chains will be sufficient
- To block all `ping` requests to our machine:

```
$ ipchains -A input -p icmp -s 0.0.0.0/0 \
> --icmp-type echo-request -j DENY
$ ipchains -L input
Chain input (policy ACCEPT):
target   prot opt     source       destination    ports
DENY     icmp ------  anywhere     anywhere       echo-request
```

- To block outgoing `ping` packets:

```
$ ipchains -A output -p icmp -d 0.0.0.0/0 \
> --icmp-type echo-request -j DENY
$ ipchains -L output
Chain output (policy ACCEPT):
target   prot opt     source       destination    ports
DENY     icmp ------  anywhere     anywhere       echo-request

$ ping -c1 landlord
PING landlord.gbdirect.co.uk (192.168.0.129): 56 data bytes
ping: sendto: Operation not permitted
ping: wrote landlord.gbdirect.co.uk 64 chars, ret=-1

--- landlord.gbdirect.co.uk ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

- Very simple examples but they show the theory

16.16 Removing Rules

- Rules can be removed by *number*, e.g. to delete the first rule in the *input* chain:

```
$ ipchains -D input 1
```

- or definition, e.g. delete the *first* matching rule:

```
$ ipchains -D output -p icmp -d 0.0.0.0/0 --icmp-type
  echo-request -j DENY
```

- To clear an entire chain use:

```
$ ipchains -F chainname
```

- If no `chainname` is given, it clears all chains

16.17 Implementing ipchains

- The rules are normally set up in the machines ‘init scripts’
- Typically by creating a script in `init.d` that is run just before networking starts
  - ♢ Example in section 16.19
- Ensure you flush existing rules first (just in case):

```
$ ipchains -F
```

- Generally start with the DENY rules then add what you want
- Maximum security

16.18 Save and restore

- Often useful to create a firewalling *‘config file’*
- `ipchains-save` outputs a text file you can store

```
[Setup firewall rules as you want]
$ ipchains-save > /etc/ip.rules
Saving ‘input’.
Saving ‘forward’.
Saving ‘output’.
$
```

- Can reinitialise your firewalling with `ipchains-restore` and your ‘config file’, e.g.

```
$ ipchains-restore < /etc/ip.rules
$
```

- Usually done in a startup script

16.19 `ipchains` setup script

- A sample script may look like:

```
#! /bin/sh
# Script to control packet filtering.
# If no rules, do nothing.
# Altered from the ipchains HOWTO
[ -f /etc/ipchains.rules ] || exit 0
case "$1" in
    start)
        echo -n "Turning on packet filtering:"
        /sbin/ipchains-restore < /etc/ipchains.rules || exit 1
        echo "."
        ;;
    stop)
        echo -n "Turning off packet filtering:"
        /sbin/ipchains -X
        /sbin/ipchains -F
        /sbin/ipchains -P input ACCEPT
        /sbin/ipchains -P output ACCEPT
        /sbin/ipchains -P forward ACCEPT
        echo "."
        ;;
  restart)
$0 stop
$0 start
;;
    *)
        echo "Usage: $0 {start|stop|restart}"
        exit 1
        ;;
esac
exit 0
```

16.20 Real World `ipchains`

- Connect out to a host but not in

```
$ ipchains -A input -d 192.168.0.131/32 -p TCP -y -j DENY
```

- `-y` limits matching to packets with the SYN bit set
  - ♢ Used when establishing connections
- No-one can open a connection from `192.168.0.131`
  - ♢ Can still connect to it from here . . .

16.21 Interface Configuration and Management

- An interface is a point of connection to a network
- Usually a single device
  - ♢ Network card
  - ♢ PPP link
- A device can have more than one interface
  - ♢ Referred to as ‘aliases’
  - ♢ Commonly used for virtual web sites

16.22 Point-and-Click Interface Administration

- Number of ways to add/edit interface details
  - ◊ Linuxconf
  - ◊ Redhat `control-panel`
  - ◊ *'By hand!'*
- For most cases you can probably use one of the two graphical methods
- Useful to understand the configuration files behind it all

16.23 `/etc/sysconfig/network-scripts`

- Directory containing scripts and config files [3]
- `ifup` & `ifdown` activate/deactivate an interface
- Argument specifies interface to act on, e.g.

  ```
  $ ifdown eth0
  ```

- `ifcfg-eth*` are config files for each interface
  - ◇ Should be numbered sequentially from 0
    - ∗ `ifcfg-eth0` is the first interface
  - ◇ Files ending in `:n` (where `n` is a number) are aliases
    - ∗ `ifcfg-eth0:0` is the first alias for the first interface

[3]This applies to RedHat only, you should see section 16.27 for information on other distributions

16.24 `ifcfg-ethx`

- Describes characteristics of a given interface
  - ♢ What device it should be known as (`DEVICE`)
  - ♢ IP address, network, and netmask (`IPADDR`, `NETWORK`, `NETMASK`)
  - ♢ Whether it is activated at boot time (`ONBOOT`)
  - ♢ Whether it can be controlled by normal users (`USERCTL`)
- Example:

```
DEVICE=eth0
IPADDR=192.168.0.129
NETMASK=255.255.255.0
NETWORK=192.168.0.0
BROADCAST=192.168.0.255
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

16.25 Altering An Interface

- It is perfectly allowable to alter interfaces while the system is running
- Requires only minimal disruption to network connectivity
  - ♢ Not a reboot
- Two simple steps
  1. Make alterations (by hand or through GUI)
  2. Restart networking
- Networking is just another service
  - ♢ `/etc/rc.d/init.d/network restart`
  - ♢ `/etc/init.d/network restart`

16.26 Adding an Interface

- Adding an alias is even easier!
  1. Add the alias
  2. Activate it
- Example: Add the following to `/etc/sysconfig/network-scripts/ifcfg-eth0:0`

```
DEVICE=eth0:0
USERCTL=no
ONBOOT=yes
BOOTPROTO=none
BROADCAST=192.168.0.255
NETWORK=192.168.0.0
NETMASK=255.255.255.0
IPADDR=192.168.0.141
```

- Then execute

```
$ ifup eth0:0
```

16.27 The 'Proper' Way

- Previous examples use scripts (not always provided)
- You can do everything manually
- Add an alias :

  `/sbin/ifconfig eth0:0 192.168.0.128`
- Check with `ifconfig` that it succeeded
- Setup routing to that interface:

  `/sbin/route add -host 192.168.0.128 dev eth0:0`
- Removing an alias:
  - ♢ `/sbin/ifconfig eth0:0 down`
  - ♢ `/sbin/route del 192.168.0.128`
- Adding an interface is similar . . .
- Probably want to add a route to the entire network not just the host

  `/sbin/route add -net 192.168.0.0 netmask 255.255.255.0 dev eth1`

16.28 Drivers

- Network drivers invariably handled by kernel modules
  - ◊ PCI NE2000 card handled by `ne2k-pci.o`
- Kernel cannot tell which module should be used by which interface
  - ◊ module loader uses lines from `/etc/conf.modules`, e.g

```
alias eth0 ne
alias eth1 ne
options ne io=0x320,0x340 irq=2,12
```

  - ◊ Above says that interfaces `eth0` and `eth1` handled by `ne` module (NE2000 ISA)
  - ◊ Options line is module-specific; permits port/IRQ specification if not autodetected

16.29 The Secure Shell in Practice (`ssh`)

- How you use `ssh` varies across systems
- Some require stricter authentication than others
- For example, within a secure environment it may not require a password
  - ♢ Works on ‘trusted host’ concept
  - ♢ Better than `rsh` due to server key authentication
- Can often be used as a drop-in replacement for `rsh` or `telnet`
- Has numerous advantages . . .
  - ♢ Sets up forwarding of X connections
  - ♢ Can compress the data sent

16.30 Secure Copying in Practice (`scp`)

- Replacement for `rcp`
- More secure
  - ♢ Encrypts all traffic
  - ♢ Uses same authentication as `ssh`
- Can copy local to remote, remote to local or remote to remote
- Example:

```
$ scp localfilename user@remotehost:remotefilename
```

16.31 Summary

- Wide range of network utilities available
  - ♢ Both maintenance and user-orientated
- *Very* flexible system
  - ♢ Can be hard to setup/maintain
- Pros outweigh cons
- Common jobs become second nature

16.32 Exercises

1. Network tools
   (a) Use `netstat -rn` to investigate the routes on your network. Explain each line of entry to a colleague.
   (b) Read the man page for `tcpdump`. Use it to monitor traffic on your host's network interface whilst other hosts are pinging each other.
2. `ipchains`
   (a) Use ipchains to set up the following configurations. In each case you should first set up the system by hand, check it. Then set it up so that the firewall rules are in place when the machine reboots.
      i. Block all incoming ICMP packets
      ii. Block only incoming ICMP 'echo-request' packets
      iii. Block all incoming telnet connections
      iv. Block *all* telnet connections
      v. Block all outgoing web requests (Port 80)
3. Network configuration
   (a) Using one of the admin tools (`linuxconf` or `control-panel` etc.) add an alias on your network interface so that your host can masquerade as some other host. DO NOT DO THIS IF YOU ARE NOT SURE YOU ARE USING A SPARE IP ADDRESS. Investigate what `ifconfig` and `netstat -rn` now report. Check that you can ping the alias from another host on the network.
   (b) If possible, fit an extra network card to one of the hosts (host b) and configure it to be on a different network. Check it can be pinged from its own host. Go to another host (host a) on the original network and add a route to host b's new interface, using as a gateway host b's original network interface. Check that you can ping it and then use `traceroute` to see the path taken by packets. Host b will have to have IPV4 forwarding enabled for this to work. Ask the tutor about which machine will be set up for this.

16.33 Solutions

1. (a) If you don't understand the output check section 16.2 or the `netstat` manpage
   (b) `tcpdump -i eth0` should monitor all network traffic. If you want to see the traffic to a particular host use `tcpdump dst host 10.0.0.3`
2. (a) The following are the list of rules needed to satisfy each situation. You should flush the chains before each one (`ipchains -F`).
      i. `ipchains -A input -p icmp -j DENY`
      ii. `ipchains -A input -p icmp --icmp-type echo-request -j DENY`
      iii. `ipchains -A input -p tcp -d 127.0.0.1 --dport telnet -j DENY` `ipchains -A input -p tcp -d 192.168.0.131 --dport telnet -j DENY`
      iv. `ipchains -A output -p tcp -d 0/0 --dport telnet -j DENY` `ipchains -A input -p tcp -s 0/0 --dport telnet -j DENY`
      v. `ipchains -A output -p tcp -d 0/0 --dport www -j DENY`
3. (a) -
   (b) Ask the tutor for details.

Module 17

Basic Tools

Objectives

At the end of this section, you will be able to:

- Use the most frequently used Linux tools to:
  - ♢ Find files
  - ♢ Get information about commands
  - ♢ View file contents
  - ♢ Get information about files
  - ♢ Operate on file contents
  - ♢ Do simple text manipulation
  - ♢ Schedule jobs
- Combine tools to solve problems
- Understand and use the Linux printing subsystem

17.1 Introduction

The basic Linux command-line utilities dealt with here, are:

Finding files	`find` `locate`
Getting info about commands	`man`

Viewing file contents	`cat`
	`less`
	`head/tail`

Getting information about a file	`ls`
	`file`
	`wc`
	`diff`
	`cmp`

Operating on file contents	`grep`
	`sort`
	`uniq`
	`split/csplit`
	`tar`
	`gzip/bzip2`

Simple text manipulation	`tr` `expr`
Scheduling jobs	`at` `crontab`

Table 17.1: Basic Linux utilities

17.2 Using Tools

- Typical Linux systems contain over 400 command-line tools
- Tools are combined (via pipes and redirection) to solve specific problems
- Most tools have a standard syntax:

```
$ command [options] [files ... ]
```

- Some arguments must be quoted
- Standard input often read if no filename given
- Most tools can take several filename arguments
- Desktop/windowing environments may provide graphical wrappers to some tools
- Serious Linux administrators and users know the key command-lines well
- The terms ‘command’ and ‘tool’ are used interchangeably here

17.3 The On-Line Manual (`man`)

- Most commands have an associated man page
- Accessed by typing:

  `$ man` *`command`*`[s]`
- Brings up a page of information usually detailing:
  - ♢ command name, section number, description
  - ♢ syntax
  - ♢ options
  - ♢ version information
  - ♢ location of configuration files
  - ♢ other related commands
  - ♢ examples of usage
  - ♢ known bugs (if any . . .)

17.4 Finding Files the Long Way (`find`)

- `find` searches the filesystem in real time;
  - ◇ Makes disks work hard
- Can find files by name, type, size, dates, e.g
  - ◇ To find all files ending with `.jpg` under the current directory:

    ```
    find .  -name "*.jpg"
    ```
  - ◇ To find all filenames ending in `.jpg` *and* modified in the last 8 days below `/etc`

    ```
    find /etc -name "*.jpg" -mtime -8
    ```
- Tests can be combined with `-o` and negated with `!`, for example:
  - ◇ To find all filenames *not* ending in `.jpg` *or* modified in the last 8 days under `/etc`

    ```
    find .  !  -name "*.jpg" -o -mtime -8
    ```
- Can execute commands on the files it finds. The name of the file found is placed in {} [1]

```
find . -name "*.gif" -exec ls -l {} \;
```

[1] This is not a resource friendly way of doing things. `xargs` may be better

17.5 Locate Files (`locate`)

- `locate` searches a periodically-updated database of the filesystem(s)
  - ◇ Not available on all systems
  - ◇ Very fast, but DB needs regular updating, e.g.

    ```
    $ updatedb
    ```

  - ◇ Usually updated nightly automatically
- Won’t be there on a fresh install
- Won’t show files created since last database update
- Given the command ‘`locate` *`string`*’, `locate` will show all files containing *string* in their full pathname, e.g.

```
$ locate logrotate
/usr/man/man8/logrotate.8
/usr/sbin/logrotate
/etc/logrotate.d
/etc/logrotate.d/cron
/etc/logrotate.d/syslog
/etc/logrotate.d/linuxconf
/etc/logrotate.d/ftpd
/etc/logrotate.d/samba
/etc/cron.daily/logrotate
/etc/logrotate.conf
```

17.6 View and Concatenate Files (`cat`)

- Displays and/or joins (con$\boxed{\text{cat}}$enates) files
- Sends the content of named file(s) to standard output
- If no filename is given, it reads from standard input and writes to standard output
- Given more than one filename, it displays each file's contents sequentially, i.e, joins them
- Example:

```
$ cat file1 file2 file3 > all_files
```

17.7 View Large Files & Output (`less`)

- `less` displays the contents of file(s) in a controlled way on stdout
  - ♢ Usually, one page at a time
  - ♢ Like UNIX/DOS command `more`, on steroids
- You can search for patterns in the file
- It allows you to move quickly to *any* point (backwards or forwards)
- Similar usage to `more`, `vi`, and `lynx`:

Action	Keystokes
Top of page	`g < ESC-<`
Bottom of page	`G > ESC->`
Forward one screen	`f ^F ^V SPACE`
Backward one screen	`b ^B ESC-v`
Up one line	`y ^Y k ^K ^P`
Down one line	`e ^E j ^N RETURN`
pattern Search forward	`/`*pattern*
pattern Search backward	`?`*pattern*
Repeat *pattern* Search forward	`n`
Repeat *pattern* Search backward	`N`
Move to *n*th line	*n*`g`
!command	Execute the shell command with $SHELL
|Xcommand	Pipe file between current pos & mark X to shell command
v	Edit the current file with $VISUAL or $EDITOR

Table 17.2: Commands within `less`

17.8 Viewing Parts of Files (`head` and `tail`)

- `head` displays the first few lines of a file
- `tail` displays the last few lines of a file
- You can specify how many lines are displayed
  - ♢ To display only the first 4 lines:

    `$ head -4` *`filename`*
- `tail -f` often used to monitor growing files

17.9 Listing File Information (`ls`)

- Without any options, `ls` lists files in the current directory
- By default all files starting with `.` (dot) aren't shown
- The most common options to `ls` include:

Flag	Option
`-l`	*Long* (detailed) listing of file info, including: size, ownership, permissions and type
`-a`	Show *all* files, including hidden ones
`-F`	Highlight directories and executables with `/` and `@` respectively
`-R`	*Recursively* list subdirectories
`-t`	Sort list by last modification *time*
`-u`	Sort list by last access time (with `-t`)
`-X`	Sort list by file e*X*tension
`-r`	Reverse order of listing
`-d`	Show directory information not directory contents

Table 17.3: Common options to `ls`

- For example:

```
$ ls -lrt
```

  show files in reverse order based on their modification time

17.10 File Classification (`file`)

- `file` displays the type of data contained in named file(s)
- Results not always correct
- Uses list of magic numbers and keywords in `/etc/magic` to determine file type [2]
- Classifications include: executable, archive, C program, ASCII text, JPEG image . . .
- Syntax:

  `file [-vbczL] [-f` *`filename`*`] [-m` *`magicfiles`*`] file...`

[2] The magic numbers file can be `/usr/share/magic` on some systems

17.11 Count Words, Lines, Characters (`wc`)

- `wc` displays the number of lines, words,[3] and characters in a file

Flag	Option
`-l`	Only displays the number of lines
`-w`	Only displays the number of ‘words’
`-c`	Only displays the number of characters

Table 17.4: Options to the `wc` command

[3] A ‘word’, in this context, is a character string surrounded by SPACEs, TABs, NEWLINEs, or a combination of them.

17.12 Differences Between Files (`diff`)

- `diff` displays the difference between two *text* files, line-by-line
- Output from `diff` can be confusing
- For example, given the files *text1* and *text2*:

  text1:
  This is a temprary test
  to check the diff
  utility

  text2:
  This is a temporary test
  to check the diff
  utility.

1. A simple line-by-line comparison:

```
$ diff text1 text2
1c1
< This is a temprary test
---
> This is a temporary test
3c3
< utility
---
> utility.
```

2. Using the context output format (`-c`):

```
$ diff -c text1 text2
*** text1       Mon Apr 19 14:46:25 1999
--- text2       Mon Apr 19 14:46:05 1999
***************
*** 1,3 ****
! This is a temprary test
  to check the diff
! utility
--- 1,3 ----
! This is a temporary test
  to check the diff
! utility.
```

3. Using the unified output format (`-u`): (Most common)

```
$ diff -u text1 text2
--- text1       Mon Apr 19 14:46:25 1999
+++ text2       Mon Apr 19 14:46:05 1999
@@ -1,3 +1,3 @@
-This is a temprary test
+This is a temporary test
 to check the diff
-utility
+utility.
```

17.13 Compare Binary Files (`cmp`)

- Displays differences between 2 *binary* files
- Locates the byte and line number of the first difference
- Can show *all* differences if required, e.g.

  `cmp -l` *`file1 file2`*
- `cmp -s` suppresses output and returns exit status
  - ♢ `0` if the files are identical
  - ♢ `1` if the files differ
  - ♢ `2` if an error has occurred
- Often used in shell scripts

17.14 Regular Expression Searches (`grep`)

- Search for regular expressions in file(s)

  i.e "$\boxed{\text{g}}$lobally find $\boxed{\text{r}}$egular $\boxed{\text{e}}$xpressions and $\boxed{\text{p}}$rint"

- Usage:

  `grep [options] search-pattern files`

- Reads standard input if no filenames are given
- Matching lines are printed to standard output
- Popular options:

Flag	Option
`-i`	Ignore case
`-l`	List only filenames containing the expression
`-v`	Reverse sense of test, i.e. find non-matching lines
`-w`	Word search, i.e. match whole word
`-E`	Extended regular expression search (more complex patterns), `egrep` similar
`-F`	Fixed string pattern search, same as `fgrep`

Table 17.5: Popular `grep` options

17.15 `grep` examples

- Search for a user in the password file

```
$ grep lee /etc/passwd
lee:x:500:500:Lee Willis:/home/lee:/bin/bash
```

- Search for events in a log

```
$ grep connect /var/log/secure
Mar  8 14:42:53 rafters in.telnetd[579]: connect from 192.168.0.129
Mar 13 10:27:40 rafters in.telnetd[724]: connect from 192.168.0.139
Mar 13 16:54:24 rafters in.telnetd[2135]: connect from 192.168.0.139
Mar 20 10:54:38 rafters in.telnetd[816]: connect from 127.0.0.1
```

- Search for a function in source code

```
$ grep "int main" *.cpp
checkrefint.cpp:int mainbit();
checkrefint.cpp:int mainbit(){
do_maint.cpp:int maint(Form &Myform){
expire.cpp:int main(void){
maint_login.cpp:int maint_login(Form &) {
```

17.16 Sort and Merge Files (`sort`)

- `sort` and/or merge files
- Acts as a filter without file arguments
- Sorts entire lines lexically, by default
- Alternative sort orders:

Flag	Option
`-n`	Numerical order
`-r`	Reverse order

Table 17.6: Alternative `sort` orders

- Other popular options:

Flag	Option
`-b`	Blanks (TAB, SPACE) ignored
`-f`	Fold lowercase to upper before sorting
`-i`	Ignore non-printable characters
`-m`	Merge files, without checking if sorted
`-t`*x*	Set field delimiter in file as *x*
`-u`	Unique, outputs repeat lines once only
`+`*POS1* `[-`*POS2*`]`	Specify a field in each line as a sorting key, starting at *POS1* and ending at *POS2* (or NEWLINE). Fields and character positions are numbered starting at 0
`-k` *POS1[,POS2]*	Alternative syntax for specifying sorting keys. Positions are numbered starting at 1

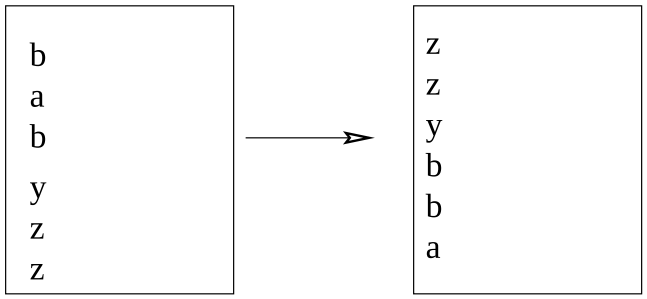
Table 17.7: Popular `sort` options

17.17 `sort` Examples

Consider `/etc/passwd` which typically contains lines in the following format:

```
username:password:UID:GID:Realname:Homedirectory:shell
```

- To sort by username:
  ```
  $ sort -t:  -f +0 -1 /etc/passwd
  ```
- To sort numerically by user ID:
  ```
  $ sort -t:  -n +2 -3 /etc/passwd
  ```
- To sort by real name within group ID:
  ```
  $ sort -t:  +3n -4 +4f -5 /etc/passwd
  ```



`sort`

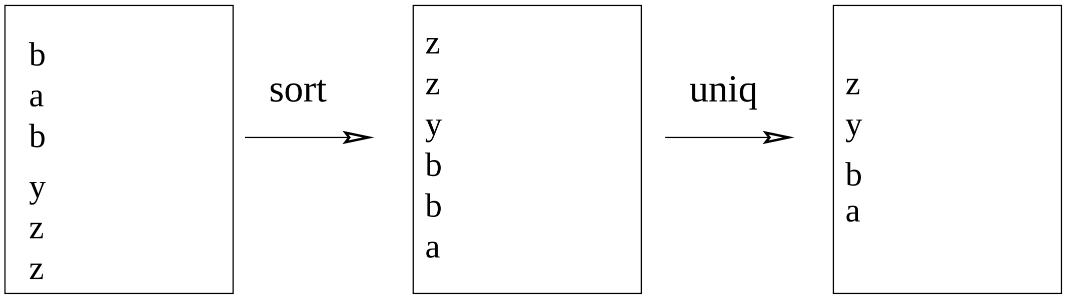
17.18 Display Unique Lines (`uniq`)

- Removes all but one of successively repeated lines
- Acts on standard input, often piped from `sort`
- Most popular options:

Flag	Option
`-c`	Count duplications and prepend number to each output line
`-d`	Duplicated lines only are displayed
`-u`	Unique lines only are displayed
`-`*n*	Ignore the first *n* fields
`+`*n*	Ignore the first *n* characters
`-w`	Specify the number of chars to compare

Table 17.8: Popular `uniq` options

- Example:



17.19 Split Files (`split`)

- Split a file into pieces
- Outputs sections to new files or standard output
- Creates files named `prefixaa`, `prefixab`, `prefixac` ...
- Main `split` options:

Flag	Option
`-l`*n*	Put *n* lines of the input file into each output file
`-b`*n*	Put *n* bytes of the input file into each output file
`-C`*n*	Put as many complete lines of the input file as is possible into the output file, up to *n* bytes

Table 17.9: Main `split` options

17.20 Splitting Files by Context (`csplit`)

- Splits file into sections determined by context (patterns or regular extressions)
- Syntax:

```
csplit [-f prefix] [-b suffix] [-n digits] filename pattern...
```

- Main `csplit` arguments:

Argument	Instruction
/regexp/[offset]	Split the file at occurrence of *regexp*. The line after the optional offset ('+' or '-' followed by a number) begins next bit of input
{repeat-count}	Repeat the previous pattern split *n* times. Substitute an asterisk for *n* to repeat until the input is exausted
`-f`*string*	Use *string* as prefix of output filename
`-b`*string*	Use *string* as suffix of output filename
`-n`*n*	Use output filenames *n* digits long

Table 17.10: Main `csplit` arguments

17.21 Compression Utilities (`gzip`)

- Linux has many compression utilities; `gzip` dominates thanks to integration with `tar`
- Takes files or standard input and compress them to file(s) or standard output
- Uses *lossless compression*, so safe on any file
- Key `gzip` options:

Flag	Option
`-r`	Recursive compression of subdirectories
`-d`	Decompress (same as `gunzip`)
`-`*`[1-9]`*	Fast or best compression, where 1 is fastest and 9 is most intense compression

Table 17.11: Key `gzip` options

- `bzip2` has better compression ratios but is not yet battle-tested or fully portable to non-Linux environments

17.22 Store and Retrieve Archives (`tar`)

- Originally designed to make tape archives
- Takes a group of files and creates one big file containing their contents *and* details
- Widely used for:
  - ♢ Compression with `gzip` [4]
  - ♢ Maintaining Linux file details (permissions, dates, ownership etc) on inferior filesystems
  - ♢ Bundling file trees for distribution
- Key *tar* options:

Flag	Option
`-c`	Create a new archive
`-r`	Append files to an existing archive
`-x`	Extract the contents from an archive
`-z`	Create/Open gzip compressed `tar` file(s)
`-f`	Filename of the file or device to hold the archive
`-P`	Pathnames are absolute

Table 17.12: Key `tar` options

- To create a `gzip` compressed archive of `/etc`:

```
$ tar -czf /home/username/filename.tar.gz /etc
```

- To extract the file in the current directory:

```
$ tar -xzf filename.tar.gz
```

[4]The `gzip` compression option is not available on all UNIX versions of `tar`.

17.23 Translating Characters (`tr`)

- Translate characters in standard input into different characters in output
- Syntax:

  `tr [options] [`*string1* `[`*string2*`]]`
- Characters in *string1* are replaced by the corresponding character in *string2*

```
$ tr ‘abc‘ ‘123‘
abcdad
123d1d
```

- Character position in both strings matters
- Both strings should be the same length [5]

[5] If string1 is shorter than string2, the extra characters at the end of string2 are ignored. If string1 is longer, GNU `tr` follows BSD in padding string2 to the length of string1, by repeating the last character. With the `-t` option it follows AT&T by truncating string1 to the length of string2.

17.24 Examples of `tr` Usage

- To replace all vowels in the input with spaces

```
$ tr 'aeiou' '     '
```

- Using character ranges to translate all lower case letters into their upper case equivalents

```
$ tr '[a-z]' '[A-Z]'
```

- Using the asterisk `*` to replace all 10 digits with the letter same 'n'

```
$ tr '[0-9]' '[n*10]'
```

- Use the `-c` option (complement) to replace all characters in *string1* which *don't* belong in a range

```
$ tr -cs '[a-zA-Z0-9]' '[\n*]'
```

N.B. This puts every word on a line by itself, by converting all non-alphanumeric characters to newlines, then 'squeezing' repeated newlines (with `-s`) into a single newline.

17.25 Execute programs at specified times (`at`)

- `at` executes a shell script at a specified time
- Syntax:

  `$ at [`*options*`]` *time* `[`*date*`]` `+`*increment*

- `stdin` is scanned for commands to execute, e.g.

```
at 2300
at> fetchmail
<CTRL>+D
```

- Some more a examples of how to specify time and date:

```
at 11pm
at 1am
at 5am tomorrow
at 08:00 Sep 12
at now
at now + 4 hours
at now + 1 day
at now + 2 months
at now + 3 years
at 5:30pm Thursday
at teatime
```

- Commands are executed in the current environment at the given time
- `stdout` and `stderr` are sent as mail

17.26 Options and commands related to `at`

- `at` belongs to a family of utilities for managing time-specified commands

Command	Purpose
`atq`	Display list of queued `at` commands
`atrm`	Remove queued `at` commands
`batch`	Schedule jobs at low CPU loading

- All of these can be run as `at` options:

Option	Purpose
`-l`	Display list of queued `at` commands
`-d`	Remove queued `at` commands
`-b`	Schedule jobs at low CPU loading
`-f`*`file`*	Specify script file in command-line
`-m`	Send mail after running `at`, whatever the `stdout` or `stderr`

- The use of `at` is controlled by `/etc/at.allow` and `/etc/at.deny` [6]

[6] UNIX NOTE: On System V these live in `/usr/lib/cron/`

17.27 Running commands regularly (`crontab`)

- `crontab` lets you submit job lists at regular times using the `crond` daemon
- 2 syntax formats:

```
$ crontab filename
$ crontab [-u username] [options]
```

- Options, etc:

Command	Purpose
`crontab` *myfile*	Install contents of *myfile* (`stdin` if no file specified) in appropriate directory
`crontab -r`	Remove the crontab for the current user
`crontab -l`	List (on `stdout`) current user's `crontab`. (might be useful for editing a cron table)
`crontab -e`	Run a text editor on your crontab file

Table 17.13: `crontab` usage

- Examples of Crontab Entries

```
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
# LW poll client1  for mail
0 10,12,16,18 * * 1-5 root /www/bin/client1_poll
LW 16/09/1998 - Hylafax cron stuff ...
0 0 * * *  root /usr/local/sbin/faxqclean
10 0 * * * fax /usr/local/sbin/faxcron -info 7
[0-59/10] 9-18 * * 1-5 root /usr/local/bin/domail
```

17.28 Evaluate expressions (`expr`)

- `expr` is used to evaluate expressions
- Takes arguments and operators on the command line
- Prints the result
- Returns zero or non zero depending on the result; can be tested with shell
- Watch out for special meaning to shell of characters like * and < or >, e.g

```
$ expr 1 + 2
3
$ expr 6 \* 7 #must escape the '*'
42
$ if expr 1+2 != 3 >/dev/null
> then
> echo yes
> else
> echo no
> fi
no
```

- Has some string manipulation facilities too

```
$ expr index abcdefg d
4
$ expr substr abcdefghijk 3 3
cde
```

- Sometimes used in shell scripts for looping

```
i=0; while expr $i != 20 >/dev/null
do echo $i; i=`expr $i + 1`; done
```

17.29 Linux Printing

- Completely network-oriented
- Any printer can be made available to any client (machine and application)
- All print jobs are sent to a queue
- Queues can be viewed, edited, maintained from anywhere
  - ♢ Subject to permission
- Formatted files can be sent straight to queues

  i.e. no ‘device drivers’
- Printer configuration via text file `/etc/printcap`, see `man -S5 printcap`

17.30 Printing documents

- Printing may be ‘dumb’
  - ♢ Data dumped straight to printer
- You get *BAD* results if formatting is wrong
- Your setup may be smart
  - ♢ Autodetect data formats and convert
- Older UNIX mainly dumb
- Modern Linux pretty smart - selects filters and transforms data streams if possible

17.31 Main Printing Tools

- `lpr` sends job to the queue for a named printer
- `lpq` returns info about jobs in a queue
- `lprm` removes unwanted jobs from a queue
- `lpc` enables system administrator to control the operation of the printing system
  - ♢ see `man lpc` for details
- Desktop environments may offer “drag ‘n’ drop”, visual facilities, etc

17.32 Using `lpr`

- Syntax:

```
lpr [options] file ...
```

- Main Options:

Flag	Options
`-P`	Name of the printer to send the job to
`-`*n*	Print *n* copies of the document
`-m`	Send mail on completion

Table 17.14: Main `lpr` options

- Example: [7]

```
$ lpr -Pdjrmt -2 filetypes.txt
```

[7]The *multiple copies* feature of `lpr` was broken in several versions, you should check you have an up to date copy before relying on this feature

17.33 Using `lpq`

- Syntax:

```
lpq [options]
```

- Options:

Flag	Options
`-P`	Name of the printer/queue to interrogate
`-l`	Get info on each file within a job

Table 17.15: `lpq` options

- Example:

```
$ lpq -Pdjrmt -l
```

17.34 Using `lprm`

- Syntax:

```
lprm [options]
```

- Options:

Flag	Options
`-P`	Remove jobs from named printer/queue
`-`	Remove all jobs belonging to yourself
user	Remove all jobs belonging to *user*
n	Remove job number *n*

Table 17.16: `lprm` options

- Example:

```
$ lprm -Pdjrmt davef
```

17.35 Basic Tools Exercises

1. *Find and Locate Files*
   Using `find` or `locate`
   (a) Display all the filenames under `/usr/sbin`.
   (b) Display all the filenames under `/usr/sbin` begining with a lowercase '`c`'.
   (c) Repeat the previous question, but *translate* the output to uppercase.
   (d) Display all the files under `/usr/sbin` which are over 5k in size in uppercase.
2. *Display Parts of Files*
   (a) Display the first 10 lines of the file `/etc/mime.types`
   (b) Display the last 10 lines of `/etc/mime.types`
   (c) Display the first 25 lines of `/etc/mime.types`
   (d) Display `/etc/mime.types` one screen at a time
   (e) While viewing `/etc/mime.types` page-by-page, search for 'html'
3. *Classify, Count and Compare Files*
   (a) Find out what file types you have in the following directories:
      i. `/etc`
      ii. `/usr/bin`
   (b) Repeat the previous question, but this time:
      i. Re-direct `/etc` listing to new file `filetypes.txt`
      ii. Append the listing for `/usr/bin` to `filetypes.txt`
   (c) Build a tool (i.e. write a command) to find out how many files are in the `/usr/bin` directory.
   (d) Create two new files from listings of 2 user's home directories, then find the differences between them.
   (e) If you are feeling adventurous, use `diff` and `ed` to make the two files created in the last question identical. (Check `man ed`)
4. *Regular Expressions*
   Using the `filetypes.txt` file that you created before, do the following:
   (a) List all the lines that contain 'directory'
   (b) List all the lines that don't contain 'directory'.
   (c) Find out how many files *are* directories, then find out how many aren't.
   (d) Why does the following give an error message (try redirecting the output to `/dev/null` so you can see the error).
   ```
   grep ASCII text filetypes.txt
   ```
5. *Sorting*
   (a) Sort the `filetypes.txt` file into reverse alphabetical order on the first field.
   You may notice that capital and lowercase letters are sorted independently, e.g. '`A`' comes before '`a`'.

(b) Repeat the first sorting exercise but ignoring case differences

(c) Sort the `filetypes.txt` files into alphabetical order on the second field (the file type).

(d) Find out how many `English text` files are listed in the `filetypes.txt` file.

17.36 Basic Tools Solutions

1. *Find and Locate Files*

   (a) Either

   ```
   $ find /usr/sbin
   ```

   or

   ```
   $ locate /usr/sbin/*
   ```

   (b) Either

   ```
   $ find /usr/sbin/c*
   ```

   or

   ```
   $ locate /usr/sbin/c*
   ```

   (c) Either

   ```
   $ find /usr/sbin/c* | tr '[a-z]' '[A-Z]'
   ```

   or

   ```
   $ locate /usr/sbin/c* | tr '[a-z]' '[A-Z]'
   ```

   (d) `$ find /usr/sbin -size 5k | tr '[a-z]' '[A-Z]'`

2. *Display Parts of Files*

   (a) Use

   ```
   $ head /etc/mime.types
   ```

   (b) Use

   ```
   $ tail /etc/mime.types
   ```

   (c) Use

   ```
   $ head -n 25 /etc/mime.types
   ```

   (d) Use one of the following

   ```
   $ less /etc/mime.types
   ```

   or

   ```
   $ more /etc/mime.types
   ```

   (e) While in one of the above, type
   `/html <RETURN>`

3. *Classify, Count and Compare Files*

   (a) Use these commands:

      i. `$ file /etc/*`
      ii. `$ file /usr/bin/*`

   (b) Use these commands

      i. `$ file /etc/* > filetypes.txt`

ii. `$ file /usr/bin/* >> filetypes.txt`

(c) The easiest solution is

```
$ ls -l /usr/bin | wc -l
```

(d) Something like this:

```
$ ls /home/davef > test1.txt
$ ls /home/mikeb > test2.txt
$ diff test1.txt test2.txt
```

(e) This solution uses the `--ed` option to `diff`, and the `ed` line-editor.

```
$ diff --ed test1.txt test2.txt > differences.txt
$ cat >> differences.txt
w
q
<CONTROL>d
$ cat differences.txt | ed test1.txt
```

You could substitute the `sed` stream editor to pipe everything through one line.

4. *Regular Expressions*

(a) Use

```
$ grep directory filetypes.txt
```

(b) Use

```
$ grep -v directory filetypes.txt
```

(c) Use either

```
$ grep -c directory filetypes.txt
```

```
$ grep directory filetypes.txt | wc -l
```

then

```
$ grep -cv directory filetypes.txt
```

```
$ grep -v directory filetypes.txt | wc -l
```

(d) Without escaping the space between *ASCII* and *text* the shell assumes the pattern has ended and takes *text* as a filename to look for; hence the error about a non-existing file. `grep` continues, however, showing lines that match "ASCII" in the files it can find.
What you probably wanted was:

```
$ grep "ASCII text" filetypes.txt
```

5. *Sorting*

(a) Use

```
$ sort -r filetypes.txt
```

(b) Use

```
$ sort -fr filetypes.txt
```

(c) Presuming the second column starts at character position 25

```
$ sort -k 1.25 filetypes.txt
```

(d) `$ grep -c ‘‘English text’’ filetypes.txt`

Module 18

More Tools

Objectives

Having completed this module you should be able use the following tools appropriately:

- `top`
- `ps`
- `find`
- `vmstat`
- `free`
- `ldd`
- `uptime`
- `xargs`
- `cpio`
- `tar`
- `gzip`

18.1 Introduction

Tools covered in this module have these functions:

Command	Function
`top`	display top CPU processes
`ps`	display process status
`find`	find files in a directory hierarchy
`vmstat`	display virtual memory statistics
`free`	display free and used memory
`ldd`	display shared library dependencies
`uptime`	display system uptime
`xargs`	build and exec commands from stdin
`cpio`	copy files to and from archives
`tar`	create and extract `*.tar` archive files
`gzip`	create and extract `*.gz` archive files

Table 18.1: More tools and their functions

18.2 Displaying System Processes (`top`)

- Displays ongoing processor activity in real time [1]
- Shows processes for *all* users, unlike `ps`
- Has several modes:
  - ♢ Secure Mode, disables potentially dangerous interactive commands
  - ♢ Cummulative Mode, shows time for a process *and* its dead children
  - ♢ No-idle Mode, ignores idle or zombie processes
- Typical output may be:

```
  PID USER    PRI  NI  SIZE  RSS SHARE STAT LIB %CPU %MEM   TIME COMMAND
22594 user     10   0   736  736   556 R      0  8.2  0.5   0:00 top
    1 root      0   0   144   96    76 S      0  0.0  0.0   0:03 init
    2 root      0   0     0    0     0 SW     0  0.0  0.0   0:19 kflushd
    3 root    -12 -12     0    0     0 SW<    0  0.0  0.0   2:42 kswapd
  486 root      5   5  3160 2356   804 S N    0  0.0  1.8   0:03 mysqld
  869 root      0   0    68   12    12 S      0  0.0  0.0   0:00 mingetty
  838 www       0   0 11468 6280   488 S      0  0.0  4.9   4:14 squid
   48 root      0   0   100   80    48 S      0  0.0  0.0   0:01 kerneld
  230 root      0   0   384  372   272 S      0  0.0  0.2   0:12 syslogd
  239 root      0   0   164  120    72 S      0  0.0  0.0   0:00 klogd
  250 daemon    0   0   164  132    88 S      0  0.0  0.1   0:00 atd
  261 root      0   0   192  160   112 S      0  0.0  0.1   0:01 crond
  272 bin       0   0   244  224   168 S      0  0.0  0.1   0:00 portmap
  283 root      0   0   572  296   248 S      0  0.0  0.2   0:17 snmpd
  295 root      1   0   136   88    60 S      0  0.0  0.0   0:00 inetd
  306 root      0   0   516  488   224 S      0  0.0  0.3   0:06 named
  317 root      0   0   124   56    48 S      0  0.0  0.0   0:00 lpd
```

[1]N.B. `top` is not available on all UNIX

18.3 Options and Interactive Commands for `top`

Significant command-line options for `top` include:

Option	Function
`-d`	delay between screen updates (seconds)
`-q`	Refresh without any delay.
`-S`	Specifies cumulative mode
`-s`	Secure mode
`-i`	Non-idle mode
`-c`	Show full *command line* instead of command name

Table 18.2: Command line options for `top`

The key interactive commands are:

Command	Function
`<SPACE>`	Update display
`fF`	add and remove fields
`oO`	Change order of displayed fields
`h` or `?`	Help on interactive commands
`S`	Toggle cumulative mode
`i`	Toggle display of idle proceses
`c`	Toggle display of command name/line
`l`	Toggle display of load average
`m`	Toggle display of memory information
`t`	Toggle display of summary information
`k`	Kill a task (with any signal)
`r`	Renice a task
`P`	Sort by CPU usage
`M`	Sort by resident memory usage
`T`	Sort by time / cumulative time
`u`	Show only a specific user
`n` or `#`	Set the number of process to show
`W`	Write configuration file /.toprc
`q`	Quit

Table 18.3: Interctive commands for `top`

18.4 Reporting process status (`ps`)

- `ps` prints info about a user's processes :

```
PID   TTY STAT TIME COMMAND
22074 p0  S    0:02 Eterm -t trans
22075 p0  S    2:13 emacs -bg black
22081 p0  S    0:00 asclock
22590 p5  R    0:00 ps
```

- Unlike `jobs` which only prints info about processes belonging to the current shell
- Various display formats:

◇ Long format (`l`), e.g.

```
 FLAGS   UID   PID  PPID PRI  NI  SIZE  RSS  WCHAN   STA TTY TIME COMMAND
     0   512 19665 19653   0   0  7000 5616  12d63d  S   p2  0:19 /usr/bin/emacs
100000   512 23656 23652   0   0  1076  548  1897da  S   p5  0:00 /usr/bin/less -is
```

◇ User format (`u`), e.g.

```
USER    PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
davef 19665  0.0  2.1  7000  5616  p2 S    09:06   0:20 /usr/bin/emacs
davef 23656  0.0  0.2  1076   548  p5 S    14:36   0:00 /usr/bin/less -is
```

◇ Jobs format (`j`), e.g.

```
 PPID   PID  PGID   SID TTY TPGID  STAT  UID   TIME COMMAND
19653 19665 19665 19652  p2 19653  S     512   0:20 /usr/bin/emacs
23652 23656 23650 19679  p5 23650  S     512   0:00 /usr/bin/less -is
```

◇ Virtual Memory format (`v`), e.g.

```
  PID TTY STAT TIME  PAGEIN TSIZ DSIZ  RSS   LIM %MEM COMMAND
23656  p5 S    0:00     125   66 1009  548    xx  0.2 /usr/bin/less -is
19665  p2 S    0:21    1645  829 6170 5620    xx  2.1 /usr/bin/emacs
```

18.5 Options for Reporting process status (`ps`)

More options to `ps`:

- Show memory info (`m`)

```
  PID TTY MAJFLT MINFLT  TRS  DRS  SIZE SWAP  RSS SHRD LIB  DT COMMAND
23656  p5    125     25   72  476   548    0  548  424   0  31 /usr/bin/less -is
19665  p2   1648   2613 2120 3672  5792    0 5792 2584   0 802 /usr/bin/emacs
```

- Show 'family tree' (`f`) for processes, e.g.

```
  PID TTY STAT TIME COMMAND
19652  p2 S    0:00 /bin/login -h oakleigh gbdirect.co.uk -p
19653  p2 S    0:00  \_ -bash
19664  p2 S    3:02      \_ /home/lee/bin/emacs -f gnus-no-server
19665  p2 S    0:23      \_ /usr/bin/emacs
19668  p2 S    3:41      \_ /usr/lib/netscape/netscape-communicator
19681  p2 S    0:00          \_ (dns helper)
```

- Show environment after command line:

```
 PID TTY STAT TIME COMMAND
19653 p2 S    0:00 -bash DISPLAY=oakleigh:0.0 TERM=xterm HOME=/home/davef PATH=
```

- Sort `ps` results by specific fields: [2]

[2] You can get et a full print-out of the sorting options by giving the erroneous command string `ps -O+`

Option	Function	Option	Function
`-Ou`	user	`-Oc`	cmd
`-OU`	uid	`-Op`	pid
`-OP`	pppid	`-Ot`	tty
`-Oo` or `?`	session	`-Ok`	utime
`-OK`	stime	`-Oj`	cutime
`-OJ`	cstime	`-Oy`	priority
`-OT`	start_time	`-Or`	rss
`-Ov`	vsize	`-Os`	size
`-OC`	pcpu	`-OS`	share

Table 18.4: CLI options for sorting `ps` results

18.6 Finding Files using specified criteria (`find`)

- `find` searches your filesystem for files matching certain criteria
- Can match on name, owner, size, modification/access time, name
  - ◊ and many others
- Can execute commands on files it finds
- Commonly used to archive sets of files, or clear out old files

18.7 Criteria used in `find` expressions

- Basic syntax:

```
find base_directory search_criteria
```

-name string	Filename matches string (Shell metacharacters included)
-mtime value	Modification time matches value
-user UID/username	Owner matches UID or username
-size size	Size of the file matches size
-perm -/+ mode	Permissions of the file match mode
-type t	File is of type `t` (f - normal file, x - executable file etc. See man page for full details)

- The values to match are very flexible, e.g. to find all files below `/home/lee` that were last modified less than 36 [3] hours ago:

```
$ find /home/lee -mtime -1.5
```

- Find all files below current directory greater than 1000k in size and with permissions `664` (`-rw-rw-r--`)

```
$ find . -size +1000k -perm 664 -exec ls -l {} \;
```

[3] 1.5 multiplied by 24 hours

18.8 Examples of using (`find`)

- Find all files ending in `.jpg` under current dir:

  ```
  $ find .  -name "*.jpg"
  ```

- Find filenames ending in `.jpg` *and* modified in the last 8 days below `/etc`

  ```
  $ find /etc -name "*.jpg" -mtime -8
  ```

- Combine tests with `-o` and negate with `!`, e.g:
  - ♢ To find all filenames *not* ending in `.jpg` *or* modified in the last 8 days

    ```
    $ find .  !  -name "*.jpg" -o -mtime -8
    ```

- Execute commands on files found. For example, to find then gzip-compress tar files:

  ```
  $ find . -name "*.tar" -exec gzip {} \;
  ```

  i.e. found filenames substitute for `{}` above [4]

- N.B. `find` searches the filesystem in real time; making disks work hard

[4]Piping the `find` results to `xargs` (section 18.14) is a more efficent approach

18.9 Reporting virtual memory statistics (`vmstat`)

- `vmstat` is used to identify system bottlenecks
- Reports on processes, memory, paging, block IO, interrupts (*traps*), and cpu activity
- SYNTAX:

```
vmstat [-n] [-V] [delay [count]]
```

- If no `delay` specified gives averages since last reboot
  - ◇ Otherwise updates every *delay* seconds
  - ◇ Shows averages since last report
- *count* is the number of updates to give
  - ◇ If no `count` is specified, just keeps on running
- Option `-n` causes header display only once [5]

[5] Allegedly! Some versions do not implement this correctly

18.10 Output from `vmstat`

- Output has cryptic headings :

Section Head	Field	Description
`Procs`	`r`	no. of runnable processes
	`b`	no. of processes sleeping
	`w`	no. of processes swapped out but otherwise runnable
`Memory`	`swpd`	virtual memory used (kb)
	`free`	idle memory (kb)
	`buff`	memory used as buffers (kb)
`Swap`	`si`	memory swapped in from disk (kb/s)
	`so`	memory swapped out to disk (kb/s)
`IO`	`bi`	Blocks sent to a block device (blocks/s)
	`bo`	Blocks received from a block device (blocks/s)
`System`	`in`	interrupts per second, inc the clock
	`cs`	context switches per second
`CPU`	`us`	user time (as % of total CPU time)
	`sy`	system time (as % of total CPU time)
	`id`	idle time (as % of total CPU time)

Table 18.5: Field descriptions for `vmstat` output

18.11 free

- Another tool to examine memory status
- Displays in kilobytes by default

`-k`	Output in kilobytes
`-m`	Output in megabytes
`-b`	Output in bytes
`-s x`	Poll every x seconds
`-t`	Display a ‘total’ line

- Simpler alternative to `vmstat`
- May be useful if `vmstat` not available
- Quick check on overall memory usage

18.12 ldd

- `ldd` prints out library dependencies for a given executable

```
$ ldd /bin/bash
        libtermcap.so.2 => /lib/libtermcap.so.2 (0x40003000)
        libc.so.6 => /lib/libc.so.6 (0x40006000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x00000000)
```

- `/bin/bash` requires three libraries to run correctly
- Very useful debugging tool
- Also needed for setting up restricted (`chroot`'d) environments

18.13 uptime

- `uptime` prints out basic information about performace of system

```
$ uptime
 4:26pm  up 165 days, 23:23,  9 users,  load average: 0.23, 0.13, 0.10
```

- Shows :
  - ♢ Current Time
  - ♢ Time since last reboot (Days, Hours:Minutes)
  - ♢ Number of logged-in connections
  - ♢ Load average past minute, past 5 minutes and past fifteen minutes

18.14 xargs

- Constructs and executes command-lines from information given on standard input
- Commonly used in conjunction with find
- Syntax:

```
| xargs command
```

- N.B. Data is normally piped to `xargs`
- Example: Delete all files named `core` not modified in the past 8 days

```
$ find / -name "core" -mtime +8 | xargs rm -f
```

- This is preferred to

```
$ find / -name "core" -mtime +8 -exec rm -f {} \;
```

  as it spawns less processes, therefore easing the load on the machine

18.15 Options to `xargs`

`-p`	Interactive mode
`-t`	Verbose mode
`-n num`	Limit arguments used to command

- Verbose mode will print out the commands it executes
- Interactive mode prints out the command-line and awaits confirmation before executing
- `-n` tells `xargs` to use at most `num` arguments to the command you are running with `xargs`
- Example:

```
$ find / -name "core" -mtime +8 | xargs -p -n 2 rm -f
rm -f /home/lee/core /home/davef/core ?...y
rm -f /usr/local/bin/core /root/core ?...y
$
```

18.16 Positioning filenames with `xargs`

- By default `xargs` places the filenames at the end of the command you give
- If they need to be somewhere else you can use `replace`
- Put `{}` at the point that you want the filenames inserted
- Example:

```
$ find /var/log -mtime -0.5 | xargs --replace cp {} /tmp
```

18.17 cpio

- Similar to `tar`
- Creates archives of files
- Operates in *copy-in* or *copy-out* mode
  - ◇ Copy-out mode writes archives
  - ◇ Copy-in extracts from them
- Takes filelist on standard input
  - ◇ Not given on the command line

`-i`	Copy-in mode
`-o`	Copy-out mode
`-A`	Append to an archive
`-F file`	Use `file` instead of standard input/output
`-H format`	Use archive format `format`

18.18 `gzip`

- Compresses data taken from `stdin` or a file
- *'Lossless'* compression
  - ♢ Safe on any file
- Various compression levels, 1-9
  - ♢ 1 - Fast, less compression
  - ♢ 9 - Slower, more compression
- By default takes `filename` and turns it into `filename.gz`

```
$ ls foo*
foobar
$ gzip -9 foobar
$ ls foo*
foobar.gz
```

  - ♢ Replaces original file
- Can write compressed data to `stdout`

```
$ ls foo*
foobar
$ gzip -9 -c foobar > foobar.gz
$ ls foo*
foobar     foobar.gz
```

  - ♢ Leaves original file intact

18.19 Unzipping

- To unzip a file there are two methods
  - ◇ `gunzip`
  - ◇ `gzip -d`
- Really just aliases
- Uncompress all files in the current directory

```
$ gzip -d *.gz
```

- `gzip` uses Lempel-Ziv coding
- Can also unpack files created with `zip`, `compress` and `pack`

18.20 tar

- `tar` creates archives of files
- Used for transferring files between machines
  - ♢ Or from place to place
- Options:

`-x`	Extract from an archive
`-c`	Create archive
`-t`	List files in an archive
`-v`	Be verbose
`-z`	Compress/decompress archive with `gzip`
`-d`	Find differences between archive and the filesystem
`-f`	Operate on a file not a tape

- `tar` all files ending in `.tex` into a gzipped [6] archive called `alltex.tar.gz`

```
$ tar zcvf alltex.tar.gz *.tex
```

- Check which files were included

```
$ tar tvzf alltex.tar.gz
```

- Extract them again

```
$ tar xzvf alltex.tar.gz
```

[6] Integrated gzipping is not available on all versions of `tar`

18.21 Raw devices and `tar`

- Originally designed to talk to magnetic tapes
- Can still write to raw devices
- Useful to maximize space
  - ◇ Fit 1.44Mb of data on a floppy
  - ◇ Don’t need any space for filesystem information

```
$ tar zc filelist > /dev/fd0
```

- Extract it again

```
$ tar zx < /dev/fd0
```

18.22 Exercises

1. Use `top` to show the processes running on your machine.
2. Make `top` sort the list by memory usage.
3. Try killing a process, a good example would be your top process itself!
4. Use `top` to re-nice a process so that it gets more, or less CPU time.
5. Find a full list of every process on your machine and their full command name using `ps`.
6. Get the same view but tell `ps` to display the output in 'family tree' mode.
7. Request that `ps` sort it's output by system time used
8. Display all the filenames under `/usr/sbin` using `find`.
9. Display all the filenames under `/usr/sbin` begining with a lowercase '`c`'.
10. Display all the files under `/usr/sbin` which are over 5k in size in uppercase.
11. Set `vmstat` running in a spare terminal updating every 5 seconds.
12. Set up `free` in another window doing the same thing.
13. Use `ldd` to find out what libraries some common applications use:
    (a) `/sbin/halt`
    (b) `/usr/bin/gnome-session`
    (c) `/usr/bin/ee`
14. Practice using `xargs` by finding sets of files and performing simple (Non-destructive!) operations on them
    e.g.
    (a) Find all files in the files system modified in the last 24 hours and make copies of them in a directory called `backup` in your home directory
    (b) Find all files over 5000k and make copies of them in the `backup` directory
    (c) Find all files ending in `.txt` and compress them using `gzip`. *NOTE: You will need to use the* `-n` *option to* `xargs` *for this. Why?*
15. Use `cpio` and `tar` to create an archive of the files you've copied in to the `backup` directory. Which do you think is easier to use? Learn one . . . forget the other!
16. Write an archive of `/etc` to a floppy as a raw archive

18.23 Solutions

1. Simply starting `top` from the command line will show all processes
2. Hitting *M* will sort the list by memory usage
3. Find your top process in the list (It will probably be near the head of a CPU-sorted list. Press *k*. You will be prompted for a process number, give the number from the PID column adjacent to your top process. You are then prompted for a signal to send the process. The default (15) should kill `top` and return you to your shell prompt
4. Again find your top process, then press *r*. You are prompted for the PID and a nice value. Unless you are super-user you can only lower the process priority (Give it a higher nice value).
5. `$ ps auxw`
   Shows a full list of processes and doesn't truncate the command-line to fit on one output line
6. `$ ps auxwf`
   Identical to the above but show the family-tree view
7. Adding `OK` to the first command line should sort the output by system time

   ```
   $ ps auxw OK
   ```
8. `$ find /usr/sbin -type f`
9. `$ find /usr/sbin -type f -name "c*"`
10. `$ find /usr/sbin -type f -size +5k | tr [a-z] [A-Z]`
11. `$ vmstat 5`
12. `$ free -s 5`
13. (a) `$ ldd /sbin/halt`
    (b) `$ ldd /usr/bin/gnome-session`
    (c) `$ ldd /usr/bin/ee`
14. (a) `$ find / -mtime -1 | xargs --replace cp {} ~/backup`
    (b) `$ find / -size +5000k | xargs --replace cp {} ~/backup`
    (c) `$ find / -name "*.txt" | xargs -n 1 gzip`

    If we didn't use `-n 1` then `xargs` would produce command lines like

    ```
    $ gzip /etc/foo.txt /var/log/bar.txt moo.txt
    ```

    Due to the way `gzip` works this will not produce the desired effect
15. The command lines would be

    ```
    $ find ~/backup -type f | cpio -o > cpio_archive
    ```

    or

    ```
    $ tar zcvf tar_archive.tar.gz ~/backup
    ```
16. `$ tar zcv /etc > /dev/fd0`

Module 19

Introduction to Editing With `vi`

Objectives

In this section, you will learn how to:

- Use the `vi` editor to view, create and edit files
    - ♢ the `vi` screen layout
    - ♢ move round in files
    - ♢ replace, insert and change text
    - ♢ search files

19.1 Text editors under Linux

- There are a number of text-editors available
- `vi` is on virtually every Linux distribution
- Also comes with 99% of Unix systems
- Everyone should have a basic understanding
- `vi` is like Linux
  - ♢ Has some very complex and powerful functions that can make your life easier
  - ♢ However, you don't *have* to know everything; you get by knowing the basics
  - ♢ Shares key bindings with many utilities
- We'll just cover the basics here, `vi` is too big to cover everything!

19.2 `vi` and your terminal

- `vi` is fundamentally text-based
  - ♢ Graphical adaptations *are* available (`gvim`
- Needs to know your terminal's capabilities
  - ♢ May not function if your terminal is misconfigured
  - ♢ Check your TERM enviornment variable
  - ♢ Terminal capabilities are listed in `/etc/termcap`
  - ♢ Generally not an issue

19.3 vi screen layout

- Lines containing simply a ˜ show that you are past the end of the file and there is nothing here.
- The terminal's bottom line is the *status line*
  - ◇ Shows status messages
  - ◇ Where you type some commands
    (The 'ed'/'ex' command set, explained later)

```
This is a test document
Some lines of text here

One, two, three
four, five, six
~
~
~
~
~
~
"test" 5 lines, 77 characters written
```

19.4 Opening files with `vi`

- Launch `vi` by typing its name on command line
- With no arguments `vi` starts with an un-named and empty buffer
- `vi filename` opens a specific file
- If you don't have write permission on a file the status line will tell you :

```
"/etc/aliases" [readonly] 152 lines, 3215 characters
```

- If there is no such file status line will say something like :

```
"some_filename" [New File]
```

19.5 `vi` Modes

- Unlike many editors `vi` does not always insert what you type into the file
- Has several ‘modes’
  - ◊ Only one is responsible for inserting text into the current file
- `vi` has 3 modes: [1]

command mode	Moving the cursor, searching and manipulating existing text
insert mode	Entering new text
‘:’ (‘ed’) mode	File manipulation, advanced searching and substitution

- `vi` starts in command mode
- Return to command mode at any time by hitting <ESC>

[1]Some people refer to "ex" instead of "ed". They are the same thing

19.6 Saving, changing file and quitting

- When you open a file, a copy of it is opened into memory
- Any changes you make apply to this copy *only*
- File on disk only changes if you explicitly say so
- To save (or *write*) a file you must be in command-mode, then type `:w`
- Can save your file under a new name, e.g.

```
:w newfilename
```

- To quit `vi` type `:q`
- `vi` normally prompts you if you have unsaved work
- To quit without saving your work type `:q!`
- `ZZ` will save your work and then quit

19.7 Moving around in command mode

- Many ways to move around a document
- You must be in command mode for the following :
  - ♢ On ‘friendly’ terminals you can use arrow keys
  - ♢ Arrow keys are sometimes unavailable on some terminals so `vi` has some alternatives

```
          k
     h         l
          j
```

- Although ‘awkward’ at first, these make your life easier
  - ♢ Always work, regardless of system type
  - ♢ Fingers stay on the ‘home’ keys

19.8 Numeric Prefixes

- Key concept: ‘numeric prefixes’ or ‘multipliers’
  - ◇ Vastly improves the usefulness of many commands
- To supply a prefix simply type the number before the command
  - ◇ `vi` will then perform the command the specified number of times.
- Note: In subsequent examples a small box indicates the position of the cursor
- Starting with

```
The quick brown
fox jumped over
the lazy dog
```

and pressing `2l` will result in

```
The quick brown
fox jumped over
the lazy dog
```

19.9 Further Movement

- `vi` also allows movements by units other than characters.
- Moving by pages :

Key	Result
`^f`	Forward one screenful
`^b`	Back one screenful
`^u`	Forward half a screenful
`^d`	Back half a screenful

- Moving by ‘words’ :

Key	Result
`w`	Go to beginning of next word
`e`	Go to end of next word
`b`	Go to start of previous word

- For these commands punctuation is not counted as part of a word
- The commands `W`, `E` and `B` act the same but *do* include punctuation in words
- NOTE: Case is important to `vi` commands, `b` and `B` are different commands!
  - ♢ The upper and lower case versions of commands are *usually* related

19.10 Further Movement - Example

- From

  This, he $\boxed{s}$aid, is an example

Key	Result
w	This, he said$\boxed{,}$ is an example
W	This, he said, $\boxed{i}$s an example
e	This, he sai$\boxed{d}$, is an example
E	This, he said$\boxed{,}$ is an example
b	This, $\boxed{h}$e said, is an example
B	This, $\boxed{h}$e said, is an example

- As with virtually all commands these may be given a numeric prefix
- From the original start-point :

Key	Result
2w	This, he said, $\boxed{i}$s an example
2W	This, he said, is $\boxed{a}$n example
2e	This, he said$\boxed{,}$ is an example
2E	This, he said, i$\boxed{s}$ an example
2b	This$\boxed{,}$ he said, is an example
2B	$\boxed{T}$his, he said, is an example

- It’s not *necessary* to know these, but they make life a lot easier when you get used to them!

19.11 Movement by lines

- What if we want to get to the *beginning* of the next line[2]?
- Commands to move to line start/end:

Key	Result
`$`	Move to the end of the current line
`0`	Move to start of current line
`^`	Move to first character of line

- Moving to start of a previous or subsequent line

Key	Result
`+`	Move to beginning of the next line
`-`	Move to beginning of the previous line

[2]A ‘line’ is the set of characters contained between newline characters, not necessarily what appears on one line in your terminal

19.12 Movement by lines - Examples

- From:

  This, he said,
  is a $\boxed{\texttt{m}}$ost
  interesting example

Key	Result
+ <RET>	This, he said, is a most $\boxed{\texttt{i}}$nteresting example
-	$\boxed{\texttt{T}}$his, he said, is a most interesting example
0 or ^	This, he said, $\boxed{\texttt{i}}$s a most interesting example
$	This, he said, is a mos$\boxed{\texttt{t}}$ interesting example

19.13 Inserting text

- You probably want more from a text editor than the ability to move a cursor!
- At the bare minimum you need to be able to insert text into a file
- Don't worry, `vi` does this with ease
- As with everything else, though, there's more than one way
- Again, while this may seem confusing you only *need* to know the bare minimum
- *But*, the more you know, the easier your life becomes!

19.14 `i` command

- The `i` command inserts text before the cursor
- This places `vi` into 'insert' mode
- Anything you type now is treated as text to insert into the file rather than as a command
- You leave insert mode by typing `<ESC>`
- This is insertion at its simplest!
- To insert text *after* the cursor we use the `a` (append) command
- Also :

Key	Result
`A`	Append at the end of the line
`I`	Insert at the beginning of the line
`o`	Create blank line below cursor for insertion
`O`	Create blank line above cursor for insertion

- If your cursor keys work then you may move around the line while in insert mode
- You can delete characters from the current insertion using backspace

19.15 Multiple Insertion

- Insertion commands can take numeric prefixes
- The result may be surprising!
- Consider the following sequence of keypresses (from command mode) in an empty document

```
5i&nbsp;
```

- The result will be :

```
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
```

19.16 Deleting Text

- `vi` has a vast array of commands for deleting text
- The ‘odd-one-out’ is `x` which deletes the character under cursor
- The rest of the deletion commands are based-around the easy to remember `d` command
- `d` on its own does nothing
- You have to tell it how much to delete
- The amount to delete is given by the keys you used when studying movement

  Example:

Key	Result
`dw`	Delete to the beginning of the next ‘word’
`3dw`	Delete 3 ‘words’
`de`	Delete to the end of the ‘word’
`db`	Delete everything before cursor to the beginning of the ‘word’
`d$`	Delete to the end of the line
`d0`	Delete to the beginning of the line

- Two more special cases :

Key	Result
`dd`	Delete the entire line
`D`	Delete to the end of the line

19.17 Changing Text

- Now we know everything we need to know to delete text, insert new text and save changes
- `vi` however likes to give us choices!
- If we find a word that is wrong, we can delete it and insert the replacement
- We're *actually* 'changing' the word
- `vi` has a family of commands for just this, all starting with `c`
- Similar to deletion, i.e. you can use `cw` to change a word, `c$` to change to the end of the line, or `3cw` to change three words
- What actually happens is that the designated amount is deleted and you are placed in insert mode

Key	Result
`cw`	Change a word
`3cw`	Change 3 words
`c$`	Change to the end of the line
`c0`	Change to the beginning of the line

19.18 Copy and Paste

- We're still missing the ability to copy a piece of text and paste it somewhere else
- `vi` does support this, but it calls it 'yanking' and putting
- All 'yanking' commands are prefixed with a `y` and follow the same rules as before, i.e. `yw`, `y$`, `3yw`
- `yy` and `Y` yank a whole line and the rest of a line, respectively
- Paste text using `p` or `P`
  - ♢ `p` pastes text *after* the cursor
  - ♢ Uppercase `P` pastes it *before*
- Deleted text is also considered to be yanked
  - ♢ `xp` will transpose two characters

19.19 Finding your place

- You can search through a file using `/`
- You will get a `/` as a prompt on your status line
- To search for the string `exam` type

  ```
  /exam
  ```

  and press `<RETURN>`
- If `vi` found your search string it will move the screen to a relevant place and highlight it
- `n` will skip to the next occurence; `N` to the previous
- Search backwards by using `?` instead of `/`

19.20 Miscellaneous Commands

- `vi` has a number of commands that don't really fit anywhere else
- `~` toggles the case of character under cursor
- `.` repeats the last action
- `u` undoes the last action
  - ♢ Linux `vi` supports multilevel undo
  - ♢ Standard `vi` does *not*
- `J` Join the current and following line

19.21 Search and replace

- vi can also replace the words it finds
- Basic form is:

```
s/searchfor/replacewith/modifier
```

- By default it only changes one occurence per line, and only checks the current line
  - ◇ If we tag the g modifier on the end it will replace all matches on the current line
- If we use a range[3] we can search and replace a specified part of a document, e.g.
  - ◇ To search and replace from lines 10 to 15 inclusive:

    ```
    :10,15 s/foo/bar/g
    ```

  - ◇ To search and replace on the whole document

    ```
    :1,$ s/foo/bar/g
    ```

[3]Not explained here, this is an advanced topic

19.22 Regular Expressions

- Sometimes it’s desireable to search for a word ‘fuzzily’
- You may know the start of a word, or the end
  - ♢ Or both, but not the bit in the middle!
- *Regular expressions* can come in useful here
- Can be used in normal searches or ‘search and replace’ commands

19.23 Regular Expression Conventions

- Lots of things in Linux use regular expressions
  - ♢ Not all ‘exactly’ the same
  - ♢ 95% similar though
- Defines certain *special characters*

Character	Result
`.`	Match any character
`[a-z]`	Match any character in the range a to z
`*`	Match the preceeding character zero or more times
`^`	Match the beginning of a line
`$`	Match the end of a line
`\<`	Match the beginning of a word
`\>`	Match the end of a word

- Strictly speaking * can apply to more than one character
  - ♢ We won’t cover that here

19.24 Regular Expression Examples

- Suppose we want to find all words ending in `ent`
- We could read the entire document to check for `ent` by hand
  - ♢ Takes far too long
  - ♢ We'd probably still miss some
  - ♢ Easier to get the computer to do it
- We could do a search using `/ent<RET>`
  - ♢ Unfortunately that would also match words beginning with `ent` or with `ent` in the middle
- `/ent\>` will jump to the next word that ends with `ent`

19.25 Regular Expression Replacement

- We can also use regular expressions in the search section of search and replace commands, e.g.

  ```
  s/\<foo/bar/g
  ```

  will replace all occurences of `foo` at the beginning of a word with `bar`

19.26 `vi` Exercises

1. *Recognizing* `vi`
   (a) Start up `vi` with no filename to see what it looks like
   (b) Exit `vi` and then start it again with the file `/etc/passwd`
   (c) What can you tell about the file from this screen?
2. *Getting used to* `vi`
   (a) Start `vi` with the the file `/etc/passwd` again
   (b) Practise the basic movement commands on the file
   (c) Check you can use both the cursors and `hjkl` to move around
   (d) Check the other movement commands work as expected
3. *Creating with* `vi`
   (a) Start `vi` with the filename `vi_test`. This should be a new file
   (b) Insert your name into the file and then save it and leave `vi`
   (c) Open the file again and check it still contains your name
   (d) Next add some more names to the file, one on each line
   (e) Go to a name roughly half way down your list. Check you can insert a name on the line above, and on the line below
   (f) Check you can append to the end of lines and insert at the beginning of lines
4. *Movement and Multipliers*
   (a) Check you can move through your file using combinations of the movement keys and numeric prefixes.
   For example
      i. Move 3 lines down at a time
      ii. Move 2 words along
      iii. Move to the beginning of the second line below your cursor
5. *Deleting with* `vi`
   (a) Try deleting various entities (Words, lines, characters) from your file
   (b) Check that these work with the numeric prefixes
   (c) You should be able to achieve all of the following
      i. Delete a word
      ii. Delete to the end of the line
      iii. Delete to the beginning of the line
      iv. Delete the whole line
      v. Delete 2 lines at once
      vi. Delete 2 words at once (Either including or excluding punctuation)
6. *Changes with* `vi`
   (a) Repeat the exercises given for delete but do changes instead of deletions

7. *Yanking and Pasting*
   (a) Copy the first line of your file and paste it so that it becomes the last line
   (b) Paste it back at the top of the file
   (c) Place the cursor at the very beginning of the file and try the following keystrokes
      i. `yyjyyp`
      ii. `2yyp`
   (d) What was the difference and can you suggest why this may be?
   (e) Check that text deleted can be pasted back
8. *Miscellaneous*
   (a) Place the cursor at the beginning of the file and try the following command sequence:
   `yyp...`
   Explain the result
   (b) Place the cursor over a letter on the middle of a word. What happens when you type `xp`?
   (c) Join all the lines of your file into one long line. Check that the movement commands regarding lines work on *actual* lines rather than the lines as seen on your screen

19.27 `vi` Solutions

1. *Recognizing* `vi`
   (a) Check you understand where the status line is, and what the ~ characters mean
   (b) `:q` should exit `vi`. If you want to make sure you're in command mode press <ESC> first. `vi /etc/passwd` will start `vi` with `/etc/passwd` opened
   (c) `vi` should tell you that this file is read only. This is because you don't have sufficient permissions to change the file. `vi` should also tell you how many lines and characters are in the file.

2. *Getting used to* `vi`
   (a) `vi /etc/passwd`
   (b) You should be fairly comfortable with the various navigation methods such as moving left, right, up and down, to the end or beginning of the line and moving up and down by intervals of pages and half pages.

3. *Creating with* `vi`
   (a) `vi vi_test`. The status line should tell you that it is a new file and each line on the main screen should begin with a ~ indicating lack of content
   (b) To insert my_name simply type :
   `imy_name<ESC>`
   There are several ways to save and exit :
      i. `:w` followed by `:q`
      ii. `:wq`
      iii. `ZZ`
   (c) `vi vi_test` and check that the text you entered is there. If not try again.
   (d) There are several ways to do this :
      i. When inserting using `i` you may type RETURN to insert a newline character. it is possible therefore to start with the cursor at the beginning of the file and type :
      `iname1<RET>name2<RET>name3<RET>` and so on
      ii. Typing `o` or `O` will open a new line for insertion
   (e) You should check that you understand which of `o` and `O` inserts above, and which below the current line
   (f) Appending to the end of a line can be done using either:
      i. `$atext_to_append<ESC>`
      ii. `Atext_to_append<ESC>`

   Inserting at the beginning can be done using any of:
      i. `^itext_to_insert<ESC>`
      ii. `0itext_to_insert<ESC>`
      iii. `Itext_to_insert<ESC>`

4. *Movement and Multipliers*
   (a) You should practice moving around using the movement characters with the numerical prefixes

   i. `3j`, `3+`, or `3<RET>`
   ii. `3e`, or `2w`
   iii. `2+`, or `2<RET>`

5. *Deleting with* `vi`

   (a) You should make sure that the various deleting methods work as you expected. If they surprise you, try to work out how they *do* work.

   (b) Again check you understand the various possibilities.

   (c) The following represent only possible solutions:

      i. `dw`
      ii. `d$`
      iii. `d0`
      iv. `dd`
      v. `2dd`
      vi. `2dw`

6. *Changing with* `vi`

   (a) The answers for this are the same as for delete except substituting `c` for `d` in each case.

7. *Yanking and Pasting*

   (a) Move to the first line of the file and type `yy`, then move to the end of the file and type `p`

   (b) Move back to the top line of the file and type `P` which will paste it above the current line.

   (c) Check you can tell the difference between the two commands.

   (d) The ‘Yank buffer’ only holds the contents of one yank operation. Both sets of keypresses yank the line we start on and the line below. However the first does this as two seperate operations and the ‘yank buffer’ only remembers the most recent. The second example yanks two lines at once, therefore placing both in the yank buffer.

   (e) You should try `ddp` and check that the text appears after being deleted.

8. *Miscellaneous*

   (a) `.` repeats the last action. In this case it is a paste operation. It could equally well have been an insert, change word or delete operation.

   (b) The `xp` command pair is useful for transposing letters.

   (c) Starting at the top of your file pressing `J` will join the following line to the current line. Repeat this until the entire file is on one line. Pressing one of the down a line keys (Such as `j`, `+` or `<RET>` should have no effect despite the illusion that there is more than one line.

Module 20

Basic X-Windows

Objectives

- On completion, you should be able to:
  - ◊ Understand the basic concepts behind networked X windowing
  - ◊ start and stop X
  - ◊ run shells and user applications under X
  - ◊ set preferences for X
  - ◊ change window managers and desktops
  - ◊ use X over a network

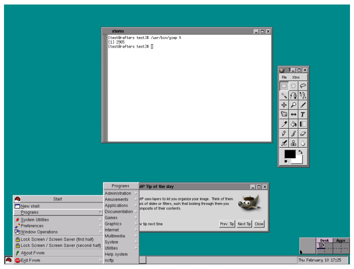
20.1 What X-Windows Is

X is a windowing system

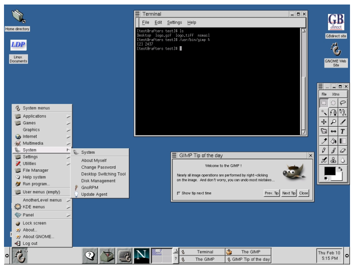
- Provides the basic graphic functions for Linux
- Designed to provide windowing to any workstation across a network, regardless of OS
- Operates on a client-server model
- Is an *application*, i.e. not a part of the OS
- The standard Linux X server is `XFree86`, commercial alternatives include:
  - ♢ Metro-X
  - ♢ Accelerated-X

20.2 X Needs Window Managers

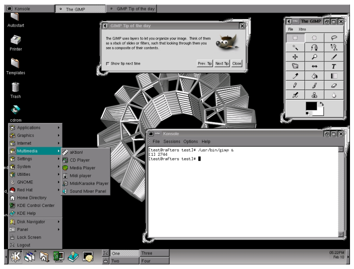
- Window managers provide the controls which allow you manipulate all graphic apps, e.g.
  - ◇ move, size and stick
  - ◇ open and close
  - ◇ maximize, minimize, iconize
  - ◇ title bars
- Determine the ‘look and feel’ of X, e.g.
  - ◇ Win95
  - ◇ Motif
  - ◇ Next Step
- Can provide ‘virtual desktops’



```
[test@rafters test]$ /usr/bin/gimp &
[1] 2965
[test@rafters test]$ 
```



```
[test@rafters test]$ ls
Desktop  logo.gif  logo.tiff  nsmail
[test@rafters test]$ /usr/bin/gimp &
[2] 2437
[test@rafters test]$ 
```



```
[test@rafters test]$ /usr/bin/gimp &
[1] 2744
[test@rafters test]$ 
```

20.3 Window Managers Are Applications

- Linux distributions contain many window managers, e.g.

Manager	Description
`fvwm2`	Motif-like look
`fvwm2`	Win95-like configuration for `fvwm2`, Red Hat default
`twm`	Bare-bones Tab WM
`olwm`	Open Look (Sun)
`olvwm`	Virtual Screen Open Look
`WindowMaker`	Next Step look, fast and lean
`enlightenment`	Gnome-compatible WM, powerful, rich, buggy
`wm2`	Ultra-lean

- Window managers are X applications, thus:
  - ◊ change manager without re-starting X
  - ◊ change X behaviour without re-start

20.4 Desktop Environments

- X + WM alone don't provide everything expected of modern desktops, e.g.
  - ♢ completely integrated drag and drop
  - ♢ universal access to a clip board
- Desktop Environments bring these facilities to Linux, bundling:
  - ♢ desktop-capable window manager
  - ♢ URL-based file manager
  - ♢ facilities to share clipboard and other data between optimized apps (inc. object linking)
- Linux currently has 3 main desktop environments:
  - ♢ CDE . . . the original commercial UNIX standard
  - ♢ KDE
  - ♢ GNOME

20.5 Starting X

- Many possibilities
  - ◇ You may be using the graphical `xdm` tool which does it for you
  - ◇ or login to command prompt then type

    ```
    $ startx
    ```

    if `.xinitrc` is not setup:

    ```
    $ xinit
    $ window manager filename
    ```

20.6 Stopping X

- Stopping:

  `<CTRL>+<ALT>+<BACKSPACE>`

- Use the window manager's menus

- If you started via `xinit`, type the following in the startup xterm:

  ```
  $ exit
  ```

- If all these fail, switch to another virtual terminal using the following keys, then kill X from the command prompt:

  `<CTRL>+<ALT>+<F1> ... <F4>`

20.7 Running Shells (Xterms) Under X

- Even under X, the most productive way to work is often via the command line (i.e. a shell)
- The standard way to access a shell prompt under X is via a terminal emulator called an `xterm`[1]
- An `xterm` shell behaves like a non-X shell, except that you can cut and paste between it and X applications
- Any number of xterms can be open at the same time
- Using `telnet` or `ssh` the `xterms` can provide shells to any number of other hosts
- To start an `xterm`:
  - ◇ From an already open xterm:

    ```
    $ xterm &
    ```
  - ◇ From a window manager menu (invariably top-level)

[1]Linux provides other terminal emulators for specialised hosts, but they are rarely necessary. There is also another category of emulators that provide advanced features such as transparent terminals etc.

20.8 Running Applications from an `xterm`

- Character-based apps:
  - ♢ Run exactly as they would outside X, unless the xterm itself has been misconfigured
- X applications:
  - ♢ Type the program's file name at the prompt: [2]

```
$ filename &
```

[2]The ampersand allows the application to run independently of the shell

20.9 Running Applications from a window manager

- Every window manager provides simple menu-based access to applications
- Application Menus are usually accessible by clicking `Mouse Button 1` on:
  - ◊ Buttons set into a task bar
  - ◊ The desktop background (root window)

20.10 Configuring X

- Default installations of Linux provide a fully functional setup for using graphic X apps
- 2 different types of X configuration that system administrators or users may need to change:
- Basic configuration of screen, mouse, keyboard behaviour, fonts
  - ♢ Could be a course in itself (classic O'Reilly manual fills a bookshelf)
  - ♢ Configuration files best edited via config tools (see next Section 20.11)
- Behaviour of desktop objects (windows, icons, taskbars, `xterms`)
  - ♢ Window manager dependent
  - ♢ Best configured via window manager preferences

20.11 Basic X Hardware Configuration

- Basic configuration for hardware is defined in the `XF86Config` file, located in `/etc/X11` [3]
- `XF86Config` is easier to edit using the following tools:
  - ◇ `XF86Setup` ... an X application which edits most basic hardware preferences (Mouse, Keyboard, Card, Monitor, Graphic Modes)
  - ◇ `xf86config` a character-based application which prompts for the same settings
  - ◇ `Xconfigurator` ... Red Hat tool sets monitor, card, screen mode, colour depth and resolution with probing
  - ◇ `mouseconfig` ... Red Hat tool sets the mouse type with probing. Useful for setting 2-button mice to emulate 3-button types by simultaneous clicking on both buttons

[3]On some older systems you may find the X configuration in `/usr/lib/X11`,`/usr/X11R6/lib/X11` or `/var/X11R6/lib/`

20.12 Basic X Software Configuration

- Under X, the user can configure every conceivable aspect of graphic display
- Users may need to change:
  - ◇ Screen font sizes, styles, familes
  - ◇ Pointer behaviour
  - ◇ Screen colours
  - ◇ Window manager
- All desktop environments and many window managers provide graphic tools for changing these configurations
- They can be set, on a system-wide or per-user basis, in the following two files:
- `.xinitrc`
  - ◇ to set the default window manager and style to be used by the `startx` command
- `.Xdefaults`
  - ◇ for fonts, pointers, colours, etc

20.13 Networked X - The Client-Server Relationship

- X works in a client-server relationship
- The client is a user application (e.g. netscape) which needs X services to display itself on a given screen
- The server is the application which provides these services, e.g. `XFree86`
- On a single-user Linux system, both apps reside on the same system
- On a networked Linux system the user can run an X application which is installed on a remote system but see it displayed on the local monitor, i.e.
  - ♢ The client application (e.g. `netscape`) is remote and the X server (e.g. `XFree86`) is local

20.14 Principles of Running Remote X Apps

- The most common use for networked X is to run client apps which are installed on remote hosts
- Reasons for running X apps on remote hosts:
  - ♢ No local installation of the app
  - ♢ Local processing or memory are insufficient
  - ♢ No local access to data

20.15 How to Run Remote X Apps

- Start the local x server:

  ```
  $ startx
  ```

- Enable (dangerous) lack of authentication

  ```
  $ xhost +
  ```

- Open a telnet connection to the remote host:

  ```
  $ telnet remote-hostname
  ```

- Set the your `$DISPLAY` *environment variable* on the remote host so that applications re-direct their graphic output to your local monitor:

  ```
  $ export DISPLAY=oakleigh:0
  ```

20.16 Authentication

- Xservers only allow authenticated hosts to connect
- On a trusted LAN you might use `xhost` in an xterm

  ```
  $ xhost +beehive
  ```
- Or edit `/etc/X0.hosts` (0 refers to display 0):

  ```
  $ cat /etc/X0.hosts
  landlord
  kebab
  samosa
  ```
- This is dangerous
  - ◇ Allows hosts to grab your mouse and keyboard
- *Only use in a trusted environment*

20.17 Better Authentication

- Can use cookie-based authentication
- Done for you if using `xdm`
- Clients look in `~/.Xauthority` for cookies to feed to server
- Server must be started with appropriate argument
  - ◇ Reads *its* `~/.Xauthority` file
- Server only looks when started
  - ◇ Too late to change once running
- Both server and clients must use the same cookies
  - ◇ Involves merging `.Xauthority` files using `xauth`
- Hard to manage - most resort to `xdm`
- Documentation is not very penetrable

20.18 Basic X Exercises

1. Figure out how to get an X session running
2. In an xterm window type `'xterm'` - what happens?
3. In an xterm window type

   ```
   export DISPLAY=xyz xterm
   ```

   what happens?
4. Start up another xterm.
   (a) Type:

   ```
   echo hello
   ```

   You should get 'hello' echoed.

   (b) Select thet `echo hello` text so that it highlights - do this by clicking the first mouse button and dragging.

   (c) Move the mouse to another xterm window; click into it to make it active if necessary.

   (d) You should be able to paste the selected text by clicking the middle mouse button (3 button mouse) or simultaneously clicking both buttons on a 2 button mouse. Try it and see.
5. Find another machine on the same network. Use `xhost` to tell it to accept connections from your machine. Start an xterm on your machine but tell it (using the `DISPLAY` variable) to display on the remote machine.

6. Go to the `/usr/X11R6/bin` directory. Try these commands:

   ```
   xeyes
   xsnow
   xfishtank
   xbill
   xdemineur
   xclock
   ```

   and any others that you like the idea of.