The Next Stage

Redirection

- You saw this briefly in the last section (*Getting Started*).

$ ***cat > myfile***

- tells *cat* to redirect all its output to file *myfile*.
- This can be used with any command, for example

$ ***cal 4 2000 > cal_file***
$ ***cat cal_file***

```
    April 2000
 S  M Tu  W Th  F  S
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
$
```

- If *cal_file* does not exist it is created.
- If it does exist ***<u>it may be overwritten</u>***

The Next Stage

Appending with redirection

- You can also append to files, for example

$ ***date >> cal_file***

- will append the output from the *date* command to *cal_file*

Error redirection

- Not all output comes out on *stdout.*
- Some errors come out on *Standard Error* (*stderr).* This appears on the screen in the same way, but comes from a different program channel.
- For example, take the following *ls* command

$ ***ls –l history_backup nofile***
nofile: No such file or directory
-r--r--r-- 1 mick firstalt 957 Oct 21 16:50 history_backup

$ ***ls –l history_backup nofile > ls_file***
nofile: No such file or directory *(this is NOT redirected)*

$ ***ls –l history_backup nofile 2> err_file***
-r--r--r-- 1 mick firstalt 957 Oct 21 16:50 history_backup

- The *2>* causes redirection of just *stderr*.

The Next Stage

Combining redirection

- In the previous example the "2>" tells the shell to redirect only the output identified as channel 2 within the command, which is usually used for error messages.
    - 1 represents the channel for normal output (stdout)
    - 2 represents the channel for error output (stderr)
- When would "1" be used? - look at the following example:-
- To redirect both *stderr* and *stdout* to the same file:-

$ ***ls -l history_backup nofile > report_file 2>&1***

- How does this work?
    - "*> report_file*" causes stdout to be redirected to the file report_file.
    - "*2>&1*" then associates channel 2 (stderr) with channel 1(stdout)
- If you do it the wrong way round:-

$ ***ls -l history_backup nofile 2>&1 > report_file***
nofile: No such file or directory

- What happened is that
    - stdout (1) is initially going to the screen
    - stderr (2) is then associated with stdout (1), thus both are now going to the screen.
    - "*> report_file*" then redirects only stdout to the file, leaving stderr still going to the screen!

The Next Stage

Input Redirection

- This is quite rare, as most programs will read a file given as an argument.
- Take the following case

$ ***cat phonelist***

- is exactly the same as

$ ***cat < phonelist***

- The < indicates input redirection.
- You can see it works, but is not necessary in this case.

- One use for input redirection is with UNIX mail

$ ***mail spot@kennel < dog.message***

- The mail program otherwise would always expect keyboard input.

The Next Stage

Exercise

- Create a file called *da* with the current date and time in it.
    * Hint - you don't need to actually type in the time and date - there's a Unix command you should be able to find
- Overwrite *da* with a calendar for July 2008- does UNIX stop you?
- Append a calendar for December 2008 to *da.*
- Append an existing file to *da. (*There are several files called f9-f14 in your home directory - they all contain ascii)
- Copy file *da* to file *new_da* using input and output redirection.
    * In other words, find a command which reads from the keyboard and writes to the screen, then modify it a) to read from a file and b) to write to a file.
- Try redirection using 2>>
- For example:-

$ ***ls -l nofile stdcode 2 >> logit***

    * What does it do?

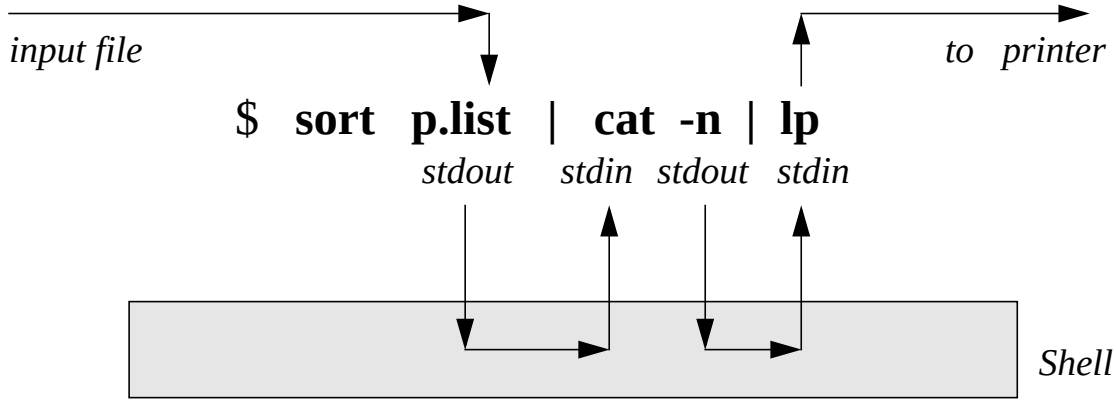
The Next Stage

Pipes

- Pipes are a facility similar to redirection, but give you the ability to pass data between processes, for example,
- Let's say we want to sort a file, place line numbers in the output, then print the result. With redirection we could:

$ ***sort p.list > sorted_list***
$ ***cat –n sorted_list > numbered_list***
$ ***lp numbered_list***

- With *pipes* we could:



- Or even combining pipes and re-direction

$ ***sort p.list | cat -n > numbered_list***

* i.e. sort *p.list*, add line numbers, then save the result in *numbered_list*.

The Next Stage

Pipes (continued)

- A VERY common pipe command:

$ ***ls –l | more***

- Piping in to *more* to automatically paginate

Tees

- You can even put a tee into the pipe!

$ ***sort p.list | cat –n | tee numbered_list | lp***

- With this command, the 'tee' passed it's stdin straight through to *stdout* - but also puts a copy in the named file
    * Result - a printout of the information (via lp) AND the information written to a disc file for later use.

- If you get confused between redirection and pipes remember
    * You PIPE to a COMMAND
    * and you REDIRECT to a FILE

The Next Stage

Exercise

- Run a long *ls* listing of the directory */usr/bin* and paginate the output (in other words display it so that it doesn't go shooting off the top of the screen).
- Use the *man* command keyword searching facility to display all man pages relating to the word *program*, one page at a time.
- If you finish quickly, take the output from the above question and display only the man pages in section 1 of the manual. Use the utility *grep* to select the lines.
- Feeling adventurous? How many lines are there in the above output?

The Next Stage

Filename Matching Characters

- In this section, we will look at special characters that are used to match file and directory names; these are often called *wild-cards.*

The character '*'

- '*' is the most commonly used character, for example:

$ ***ls *.memo***

- would list all file ending in '.memo' in the current directory
- The * matches zero or more characters
- If we had the following files in a directory:–

f f1 f23 f456 laf full iffy

ls argument	f*	*f	*f*
would match files	f f1 f23 f456 full	f laf	f f1 f23 f456 laf full iffy

- (The above files are located in your *Admin* directory if you want to try the commands.)