

Chapter 2: Using Linux at the command line

Chapter 2 Using Linux at the command line



Chapter 2 Outline

- In this chapter we will learn how to:
 - ✓ Use the linux shell productively
 - ✓ Control access to files
 - ✓ Combine key linux filter programs using pipelines
 - ✓ Navigate the filesystem and manage files
 - ✓ Edit text files with `vi`
 - ✓ Get help through on-line documentation

Basic use of the shell

- Basic use of the shell

Logging in and out

Simple commands

Command options

Command arguments

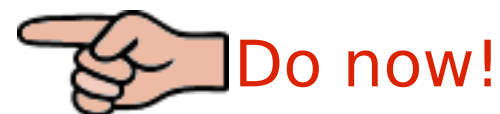
Command history

Absolute and relative
pathnames

File name completion

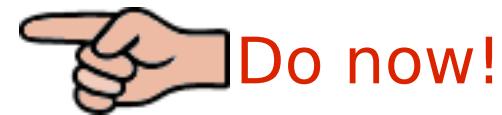
Logging in

- After booting, a desktop SuSE Linux system will usually present a graphical login dialog
 - Enter your login name and password (both are case sensitive)
 - KDE desktop environment started
- Systems (eg servers) which do not run a graphical desktop will present a command line login
 - Enter login name and password
 - A 'shell' (command interpreter) is started
- Our machines are currently configured for a command line login
 - We will reconfigure them to use a graphical login and KDE desktop later
- Please log in now:
 - Log in as the user 'tux'
 - Supply the password 'penguin'



Logging out

- To logout from a command-line environment:
 - Enter the “end of file” character (usually `^D`)
 - Or type the command `exit`
- Please:
 - Log out
 - Log back in again

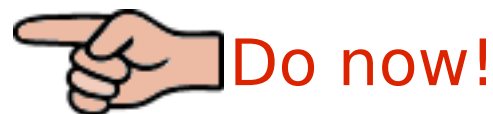


The shell

- The 'shell' is the linux command interpreter
- The shell operates in a command processing loop:
 - Displays a 'prompt' and reads a command line
 - Performs various substitutions and expansions on the command line
 - Executes the resulting command and waits for it to finish
 - Loops back and prompts for another command
- Several shells have been written for UNIX and Linux
 - Bourne shell (`sh`), Korn Shell, C Shell, Bourne Again Shell (`bash`)
 - The core feature set of all these shells is very similar
 - We will focus on `bash`, the most popular shell on Linux

Simple commands

- Try these simple commands:



```
$ hostname
```

```
snowwhite
```



Reports the name of this machine

```
$ date
```

```
Fri Apr 16 11:48:33 BST 2004
```

```
$ id
```

```
uid=500(chris) gid=100(users) groups=100(users),14(uucp)
```

```
$ cal
```

```
April 2004
```

```
Su Mo Tu We Th Fr Sa
```

```
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
```

For simplicity we show the prompt as a '\$'. The default configuration of SuSE linux uses a longer prompt which includes your login name, hostname and current directory

Command options

- Command *options* modify the behaviour of a command
 - Usually, an option is a single letter prefixed by '-'

```
$ cal -y
```

```
... calendar for the entire year ...
```

```
$ date -I
```

```
2004-04-16
```



Options are case sensitive



The date in ISO format

- Some commands also have 'long' options
 - Begin with '--'
 - Supplement or replace the traditional single character options

```
$ date --iso-8601
```

```
2004-04-16
```



Same as '-I'

Command arguments

- Most commands accept *arguments*
 - The command name, options, and arguments are separated by *whitespace* (spaces and tabs)
 - Arguments are often the names of files or directories on which to operate

```
$ cal 1955
```

```
... calendar for the year 1955 ...
```

```
$ ls /home
```

```
chris dilbert tux
```

← `ls` lists the contents of the specified directory `/home`

- Options and arguments are often used together:

```
$ ls -l /home
```

```
total 6
```

```
drwxr-xr-x  66 chris  users  3328 2004-04-16 11:48 chris
drwxr-xr-x  17 dilbert users  1112 2004-02-09 11:52 dilbert
drwxr-xr-x  18 tux    users  1240 2004-03-30 20:29 tux
```

← The `-l` option requests a 'long' listing

Command history

- `bash` remembers the most recent commands you've entered
 - stored in the file `.bash_history` in your home directory
 - survives across logout / login, shared by all instances of `bash`
 - size of history file is configurable, set to 500 commands in SuSE Linux

- The `history` command shows your command history

`history` shows your entire command history

`history 10` shows the last ten commands

`history -c` clears your command history

- Previous commands can be selected and re-executed

`!85` re-execute command 85

`!string` re-execute most recent command that began with string

`!!` re-execute last command

Command history (continued)

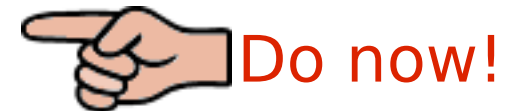
- You can also edit your command history on screen.
 - The following keys are used:

↑	scroll back through history
↓	scroll forward through history
←	move left along line
→	move right along line
<code>string</code>	insert text string into line
<code></code>	delete character
<code><ENTER></code>	execute the command

Command history (continued)

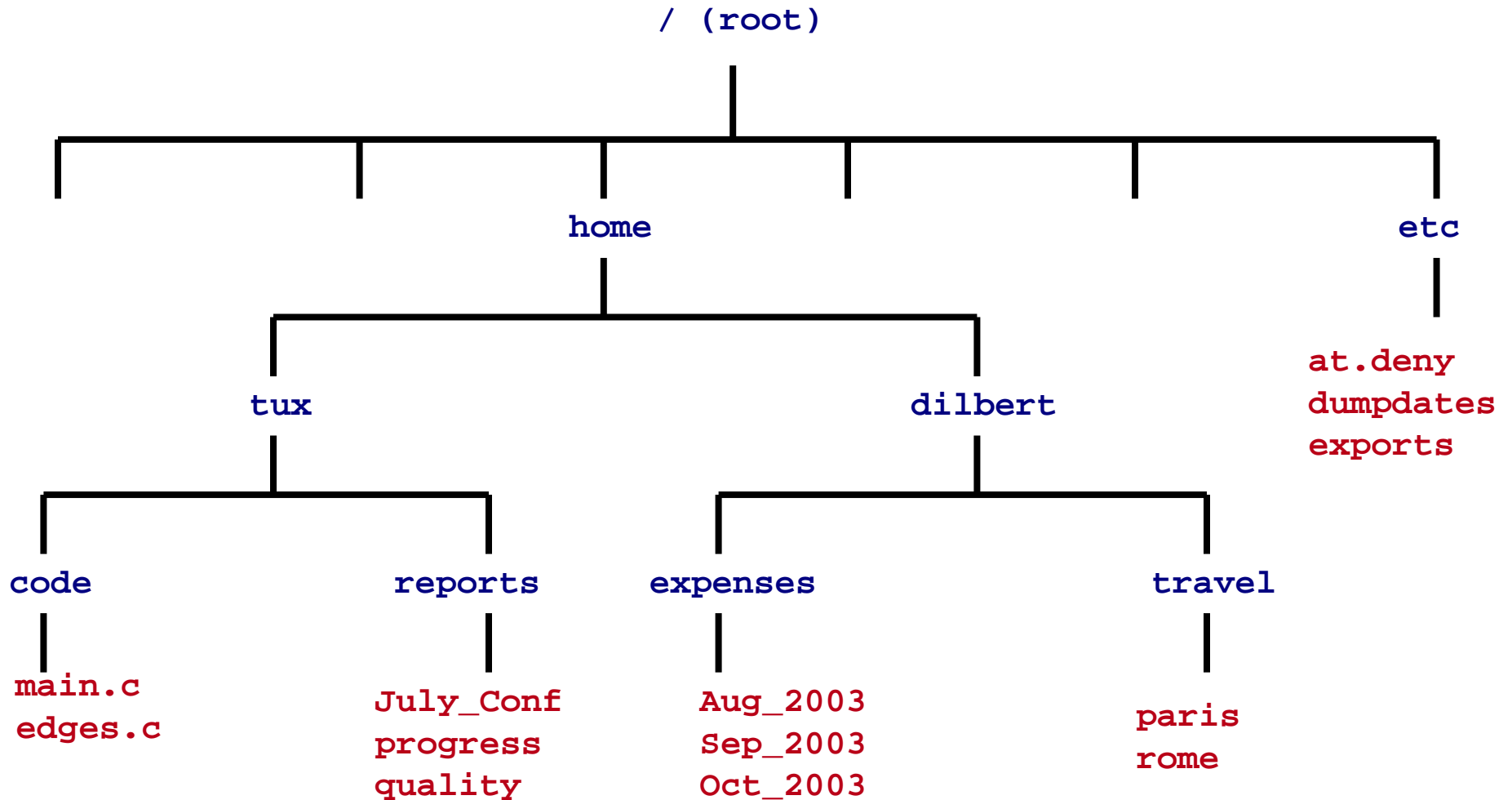
- To provide you with a command history, execute the following commands:

```
$ pwd
$ hostname
$ cal -y
$ date
$ id
$ ls /etc/hosts.allow
$ cat /etc/fstab
```



- Recall your command history with the `history` command
- Recall the `date` command by command number
- Recall the most recent command beginning with 'ho'
- Using the arrow keys, recall the 'ls' command, changing the file name from `hosts.allow` to `hosts.deny`

Absolute pathnames



Absolute pathnames (continued)

- The linux file system is organised in a tree structure
- The top level directory of the tree is called the *root directory* and is named '/'
- A file or directory can be referenced using an *absolute pathname*
 - Starts with a '/'
 - Traces a path from the root of the tree to the file
 - Uses '/' (forward slash) to separate components of the pathname

- Examples:

`/etc/at.deny`

`/home/dilbert/travel`

`/home/tux/reports/quality`

Relative pathnames

- Pathnames not beginning with '/' are relative to the current directory
- Examples (assuming `/home/tux` is the current directory):

```
reports
```

```
code/main.c
```

- Every directory has a special entry named `..` which references the *parent* directory
 - The directory immediately above it in the tree
- Use relative pathnames beginning with `..` to move outside the current directory
- Examples (assuming `/home/tux/code` is the current directory):

```
../reports/July_Conf
```

```
../../dilbert/travel/paris
```

File name completion

- When entering a command, bash will perform *filename completion*
 - Press the TAB key
 - bash will complete as much of the name as is unambiguous, based on the name completed so far and the names of the existing files
 - Less typing, less chance for typing mistakes
 - If there are several possible matches, [TAB] [TAB] will show them all

- Example ([T] denotes the TAB key)

```
$ ls -l /h[T]/t[T]/re[T]/Ju[T] ... completes to:
```

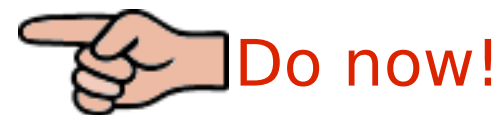
```
$ ls -l /home/tux/reports/July_Conf
```

- Using file name completion to minimise typing, do a long directory listing (`ls -l`) on these files:

```
/usr/X11R6/bin/showfont
```

```
/usr/share/xscreensaver/screensaver-diagnostic.xpm
```

```
/boot/vmlinuz.config
```

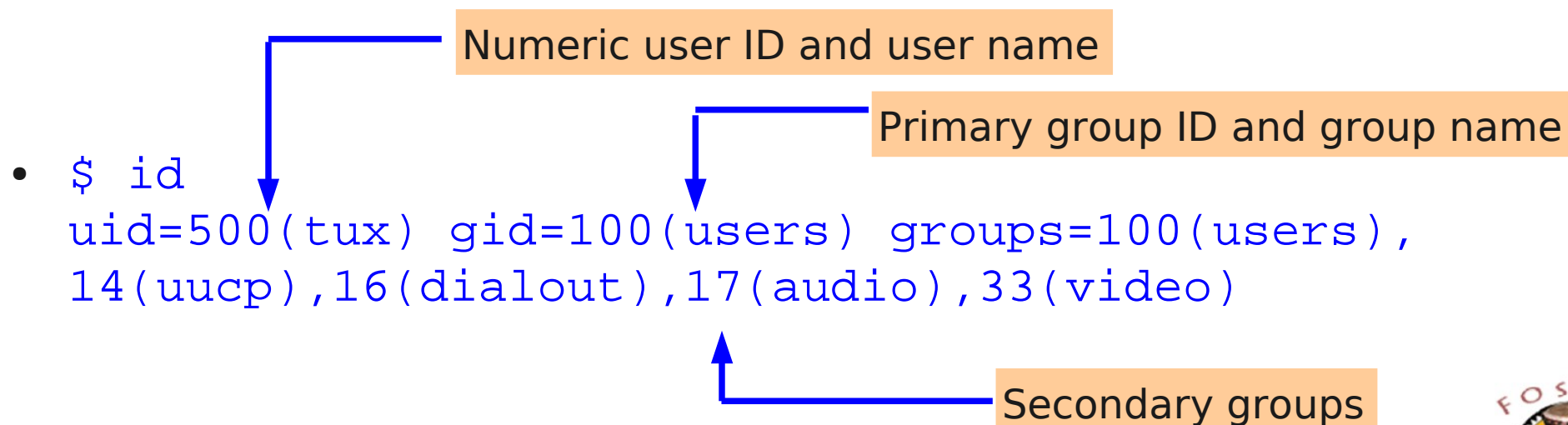


The linux security model

- The linux security model
 - Users and groups
 - The superuser
 - Standard file permissions
 - Changing access permissions
 - Representing file permissions in octal
 - setuid and setgid programs
 - Changing ownership with `chown`

Users and groups

- Every user has an account name (e.g. `tux`) along with an associated numeric user ID (e.g. 500)
- Every user is associated with one named group which is their *primary group*
 - Groups allow additional flexibility in assigning access permissions
- Users can also be associated with one or more *secondary groups*
- The command `id` shows your user identity and group memberships



The user `root`

- Linux has a privileged user account called the *super-user*
 - The account name is usually `root`
 - The numeric user ID is zero
- `root` can access or modify any file or directory and run any command
 - Only log in as `root` if you are doing something that requires it
- You can start a new shell as `root` using the `su` command


```
$ su -  
Password: suseroot  
earth: ~ #
```

The `'-'` flag causes `root`'s normal login environment to be established

You are prompted for the password
It is not echoed to the screen

The `'#'` in the prompt warns you that you are `root`

User identity and the super user

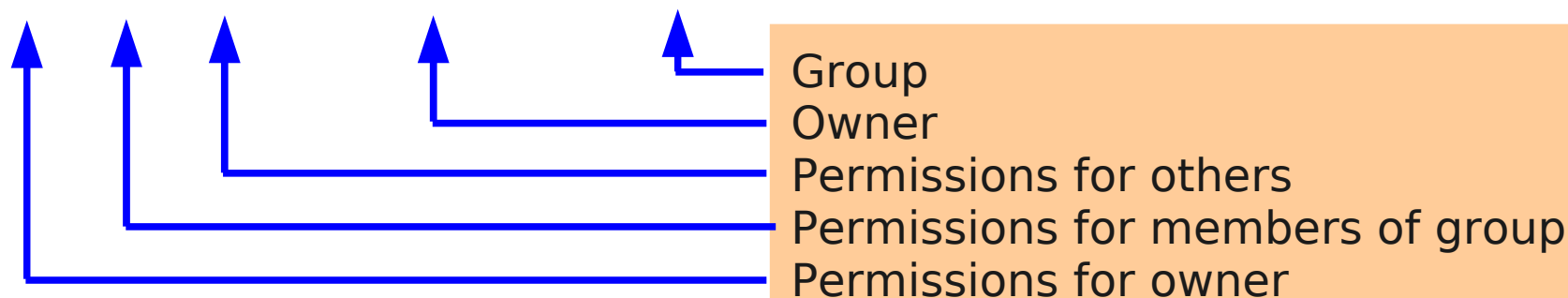
- Log in as `tux`, with password `penguin`  **Do now!**
 - What is your prompt string? _____
- Run the `id` command
 - What is your numeric user ID? _____
 - What is your primary group name? _____
 - What are the names of your secondary groups? _____
- Run the command `'su -'` to start a shell as the superuser
 - Supply the password `suseroot`
 - What is your prompt string? _____
 - What is your numeric user ID now? _____
 - What is your primary group ID? _____
- Type `exit` (or enter `^D`) to exit from your superuser shell
 - Confirm (from your prompt) that you are no longer logged in as root

Standard file permissions

- Every file and directory has a set of attributes:
 - An owner (by default, the person who created it)
 - A group (by default, the primary group of the person who created it)
 - Three sets of access permissions, relating to:
 - The owner of the file
 - Users who are members of the file's group
 - Everyone else (“others”)
- These attributes are shown in a long directory listing:

```
$ ls -l etclist
```

```
-rw-r--r--  1 tux  users 65584 2004-03-16 11:30 etclist
```



Standard file permissions (continued)

- There are three access permissions in each set
 - The meanings of these permissions differ slightly depending on whether they are applied to a regular file or a directory

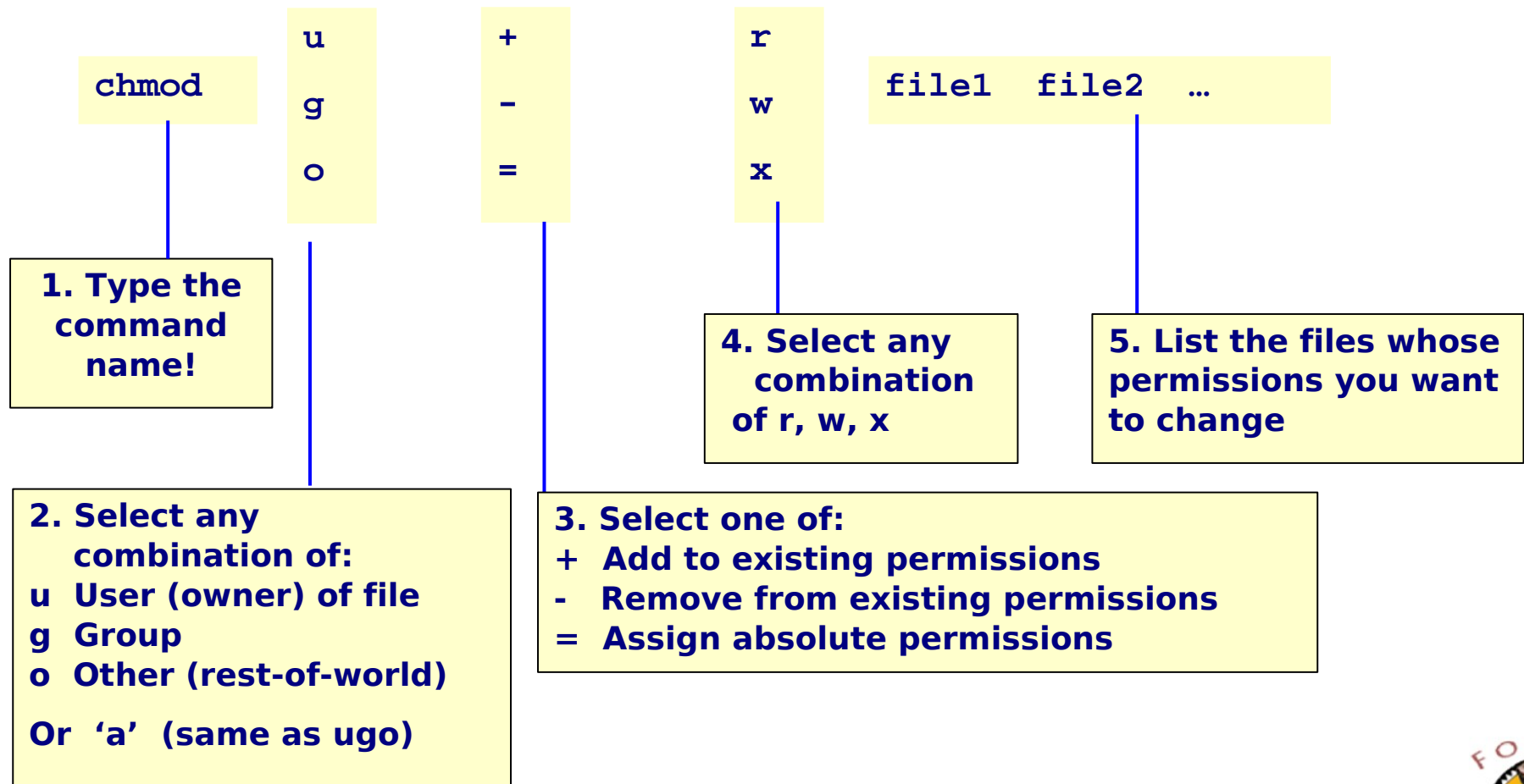
Permission	Meaning for a regular file	Meaning for a directory
r (read)	Able to see the contents of the file	Able to list the contents of the directory
w (write)	Able to change the contents of the file	Able to create or delete files or subdirectories
x (execute)	Able to run the file as a program or a script	Able to make the directory “current” or use it in a path name

The permissions are shown as a group of nine characters, for example:

rwXr-Xr-X

Changing access permissions with `chmod`

- The command `chmod` is used to change file permissions
- How to build yourself a `chmod` command in five easy lessons



Changing access permissions (continued)

- Only the owner of a file (or the superuser) can change the file's permissions

```
$ chmod u+x hello.txt
$ chmod go-w display.object
$ chmod a-wx opensource openwindows
$ chmod u=rw open*
```

- To set different access permissions for different users, either use two `chmod` commands or separate changes with a comma:

```
$ chmod u=rwx index
$ chmod go=r index
```

Or:

```
$ chmod u=rwx,go=r index
```

Question: What do you think '`chmod u=rwx go=r index`' does?

Representing file permissions in octal

- Since each of the file permissions (rwxrwxrwx) is either on or off, a file's permissions can be represented by 3 groups of 3 binary digits
 - Then each set of 3 bits can be written as an octal digit (0-7)

<code>rwx</code>	<code>rw-</code>	<code>r--</code>
<code>111</code>	<code>110</code>	<code>100</code>
<code>7</code>	<code>6</code>	<code>4</code>

- This notation can be used by `chmod`; e.g.
 - `$ chmod 644 hello.txt`
 - `$ chmod 400 hello.txt`
- Some early versions of `chmod`, and a few other commands which deal with access permissions, only understand the octal notation

Special file permissions

- In addition to the standard 9 permission bits (`rwxxrwxrwx`) there are three more bits in a file's 'mode':

Octal Value	Shown as	Name	Meaning for files	Meaning for Directories
1000	t as the 'other execute' permission	Sticky bit	Originally this bit indicated that executable programs should stay in memory after they have terminated. This meaning is now obsolete	You can only delete files if you own the file or you own the directory. Often used for communal directories such as <code>/tmp</code>
2000	s as the 'group execute' permission	Set Group ID	If the file is executed, the effective group ID is set to the group of the file	Files created in the directory belong to the directory's group and not to the primary group of the user
4000	s as the 'user execute' permission	Set User ID	If the file is executed, the effective user ID is set to the owner of the file	-

Special file permissions (continued)

- There are special notations in `chmod` to allow the special file permissions to be set or unset
- When listed using `'ls -l'` the special permissions show up as an 's' or a 't' in the execute permission positions

```
$ touch temp3
$ chmod 777 temp3
$ ls -l temp3
-rwxrwxrwx    1 chris  users    0 2004-03-16 14:56 temp3
$ chmod u+s temp3
$ ls -l temp3
-rwsrwxrwx    1 chris  users    0 2004-03-16 14:56 temp3
$ chmod g+s temp3
$ ls -l temp3
-rwsrwsrwx    1 chris  users    0 2004-03-16 14:56 temp3
$ chmod o+t temp3
$ ls -l temp3
-rwsrwsrwt    1 chris  users    0 2004-03-16 14:56 temp3
$
```

setuid and setgid programs

- The 'setuid' and 'setgid' permissions allow a user to run a command with the effective identity of the owner / group of the file
 - Grants the user different privileges for the duration of the command
- Example:
 - User's (encrypted) passwords are stored in the file `/etc/shadow`; only root has write permission on this file
 - Users can change their passwords using the command `/usr/bin/passwd`

- To allow this, the ownerships and permissions are like this:

```
$ ls -l /etc/shadow
-rw-r----- 1 root shadow 768 2004-03-05 10:03 /etc/shadow
$ ls -l /usr/bin/passwd
-rwsr-xr-x 3 root shadow 77204 2003-09-24 00:04 /usr/bin/passwd
```

- It's important to be sure that setuid programs are trustworthy, especially those owned by root

Changing ownership with `chown`

- The `chown` command can change the ownership and group of a file

```
chown owner.group file1 file2 ...
```

- Example:

```
$ chown root.wheel foo bar
```

- You can change just the owner:

```
$ chown root foo
```

- or you can change just the group:

```
$ chown .wheel bar
```

```
$ chgrp wheel bar
```

- Only `root` can change a file's owner
 - Ordinary users can change a file's group only if they are members of both the original and the new group

Exercise: File permissions and ownerships

1. You should initially be logged in as tux for this exercise
2. Using the `touch` command, create a file called `sample`

```
$ touch sample
```

- Who owns the file `sample`? _____
 - What group does the file belong to? _____
 - What are the initial access permissions on the file? _____
2. Using `chmod`, create the following sets of access permissions, in turn, on the file `sample`. After each change, verify the permissions by doing a long listing of the file

```
rw-----
```

```
rw-rw-rw-
```

```
rwxrwxrwx
```

Exercise continued

4. As the user `tux`, try to change the ownership of the file `sample` to be owned by the user `dilbert`.
 - What happens?
5. Use the `su` command to switch to a superuser shell
 - Try again to change `sample` to be owned by `dilbert` (It should work this time)
 - Change the group ownership of `sample` to the group `trusted`
6. Exit from the superuser shell
7. Do a long listing of `sample` and verify the ownership and group

End of Exercise

Filter programs

- Filter programs
 - Six useful commands
 - Standard input and output
 - Redirecting standard output
 - Redirecting standard error
 - Filter programs
 - Redirecting standard input
 - Using programs in combination
 - Pipelines

Six useful commands


- There are hundreds of command line tools for linux. There are only about 30 that you need to know to be proficient at the command line
 - We have already met `date`, `id`, `ls`, `touch`, `chmod` and `chown`
- In this section we'll meet another six useful commands:
 - `less` Browse text files
 - `grep` Search for patterns in files
 - `wc` Count characters, words, lines in a file
 - `head` Display the beginning of a file
 - `tail` Display the end of a file
 - `sort` Sort the contents of a file
- Individually, each command does a fairly simple job
 - Much of the power of the linux command line comes from using tools in combination

Browsing text files with `less`

- The program `less` (an extension of an earlier program called `more`) provides a simple way to display a text file
- `less` is an interactive program and waits for you to enter a command
 - Bidirectional scrolling
 - Searching

Command	Meaning
SPACE	Scroll forward one screen
b	Scroll backward one screen
Down arrow	Scroll forward one line
Up arrow	Scroll backward one line
/string	Search forward for string
?string	Search backward for string
n	Repeat previous search
5G	Go to line 5
h	Display help screen
q	Quit (back to command prompt)

Browsing text files with `less` (continued)

- Use `less` to browse the file `/etc/profile`  **Do now!**
 - Use the down arrow key and/or the SPACE bar to browse through the file
 - Return to the beginning of the file (i.e. go to line 1)
 - Search forward for the string 'SuSE'
 - Repeatedly search forward for further occurrences of the string
 - How many times does the string appear in the file? _____
 - Display the help screen
 - Quit from `less` back to the shell prompt

Searching for patterns with `grep`

- The program `grep` searches one or more text files for lines that match a specified pattern. At its simplest, it is used like this

```
$ grep 'string' file1 file2 ...
```

- Lines in the file(s) that contain a match for the string are displayed
 - Note: putting the string inside single quotes is not always necessary but is good practice, for reasons we will examine in chapter 8
- Examples:

```
$ grep 'tux' /etc/passwd  
tux:x:504:100:Tux Penguin:/home/tux:/bin/bash
```

```
$ grep 'Clothes' shopping  
Supermarket 50 Clothespegs 1.25  
Clothes 1 Trousers 24.99  
Clothes 6 Socks 9.00  
Clothes 2 Skirt 28.00
```

Options for `grep`

- Command line arguments for `grep` include:

Option	Meaning
-r	Recursive: search all files in and below a given directory
-i	Ignore upper / lower case distinctions
-l	Show only the names of files that contain a match; not the matching lines
-v	Display lines that do <i>not</i> contain a match for the pattern
-Cn	Show n lines of context before and after each matching line
-An	Show n lines of context after each matching line
-Bn	Show n lines of context before each matching line

Anchoring the search

- The string that `grep` looks for is called a *regular expression*
 - Can contain special characters that match specific patterns in the text
 - Not covered in depth in this course
- The characters '^' and '\$' anchor the search to the beginning and end of the line respectively

```
$ grep 'Clothes' shopping
Supermarket 50 Clothespegs 1.25
Clothes 1 Trousers 24.99
Clothes 6 Socks 9.00
Clothes 2 Skirt 28.00
$ grep '^Clothes' shopping
Clothes 1 Trousers 24.99
Clothes 6 Socks 9.00
Clothes 2 Skirt 28.00
```

•

Counting characters, words and lines with `wc`

- The command `wc` counts lines, words and characters in its input files

```
$ wc /etc/passwd shopping
 29      64    1510 /etc/passwd
 15      56     491 shopping
 44     120    2001 total
```

- Command line options include:

Option	Meaning
-l	Show only the line count
-w	Show only the word count
-c	Show only the character count
-L	Show the length of the longest line

Displaying the start of a file with head

- The `head` command displays the beginning of one or more files

```
$ head -N file1 file2 ...
```

- Displays the first *N* lines of each file (default = 10 lines)

- Example:

```
$ wc shopping
```

```
    15     56    491 shopping
```

```
$ head -4 shopping
```

```
Supermarket  1   Chicken    4.55
```

```
Supermarket 50  Clothespins  1.25
```

```
Bakers       3   Bread       2.40
```

```
DIY          1   Hosepipe   15.00
```

← The file shopping has 15 lines

← Here are the first 4

Displaying the end of a file with `tail`

- The `tail` command displays the end of a file

```
$ tail -N file1 file2 ...
```

- Displays the last N lines of each file (default = 10 lines)

- Example: the last line of the `passwd` file:

```
$ tail -1 /etc/passwd
```

```
tux:x:504:100:Tux Penguin:/home/tux:/bin/bash
```

- The `-f` option causes `tail` to wait after reaching the end of the file
 - Any text subsequently appended to the file is displayed
 - Very useful for monitoring the growth of log files

Sorting a file with `sort`

- The `sort` command sorts its input line by line
 - By default, does alphanumeric sort on entire line
- Command line options include:

Option	Meaning
<code>-f</code>	Ignore upper/lower case distinction
<code>-n</code>	Numeric sort
<code>-r</code>	Reverse sort
<code>-k N</code>	Sort on field N (first field is 1)

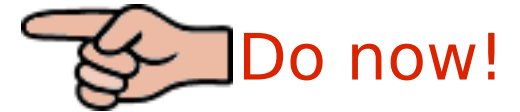
Example of using sort

```
$ sort -n -r -k 4 shopping
Clothes      2      Skirt      28.00
Clothes      1      Trousers   24.99
DIY          1      Hosepipe   15.00
DIY          20     Sandpaper  10.00
Clothes      6      Socks      9.00
DIY          2      Doorknob   8.40
Bakers       2      Quiche     6.50
Supermarket  1      Chicken    4.55
Bakers       3      Bread      2.40
DIY          2      Screwdriver 2.00
Bakers       10     Muffin     1.95
Supermarket  2      Milk       1.25
Supermarket  50     Clothespegs 1.25
DIY          50     Nails      0.95
```

Reverse numeric
sort on fourth field

More `sort` examples for you to try

- Try the following commands; make sure you understand the results



```
$ sort shopping
```

```
$ sort -r shopping
```

```
$ sort -k 3 shopping
```

```
$ sort -k 2 shopping
```

```
$ sort -n -k 2 shopping
```

```
$ sort -n -r -k 2 shopping
```

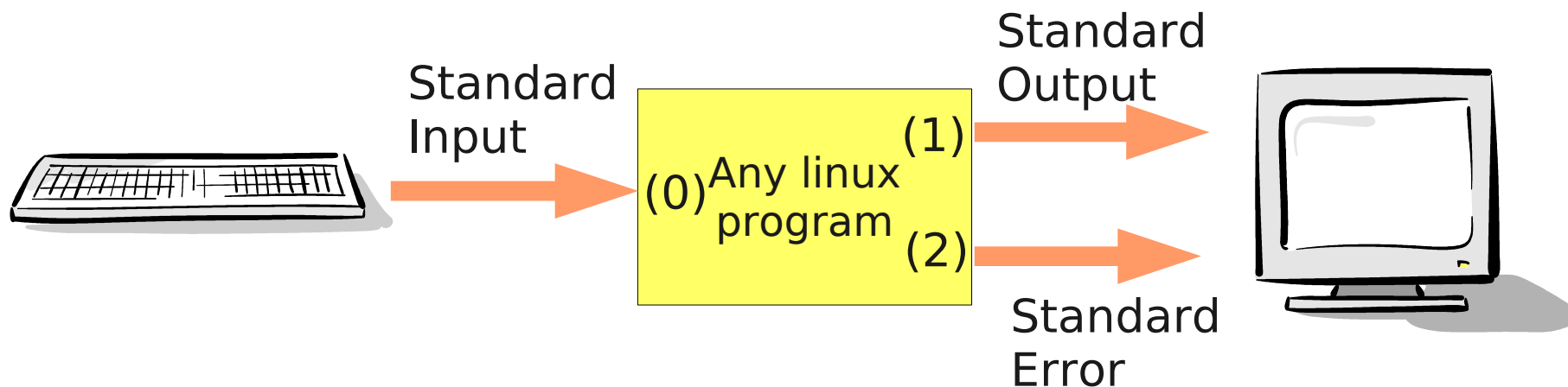
- Most commands allow you to combine multiple options, e.g.

```
$ sort -nr -k 2 shopping
```

```
$ sort -nrk 2 shopping
```

Standard input and standard output

- Every program started from the command line has three standard streams:
 - Stream 0: Standard input (stdin): from the keyboard by default
 - Stream 1: Standard output (stdout): to the terminal by default
 - Stream 2: Standard error (stderr): to the terminal by default



Redirecting standard output

- The “normal” output from a program is written to standard output
- The shell can be told to redirect standard output to a file

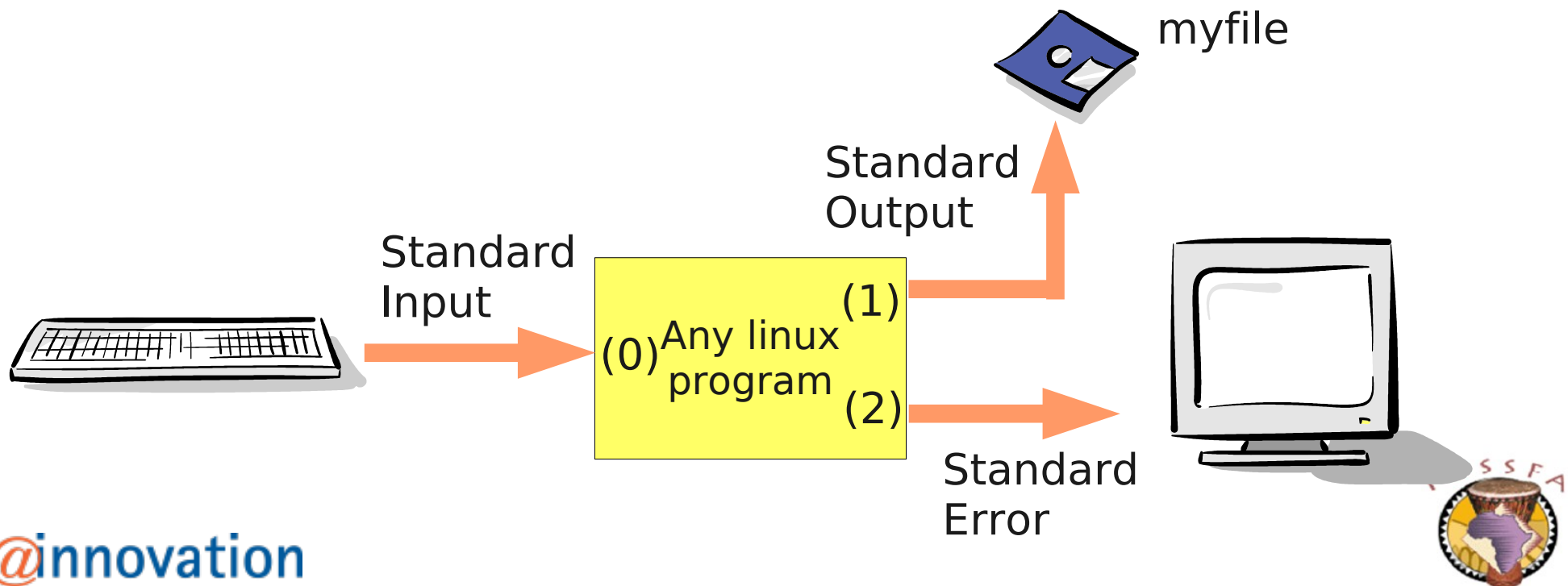
```
$ date > myfile
```

```
$ ls /opt > myfile
```

```
$ ls /boot >> myfile
```

Beware! `myfile` will be overwritten if it exists

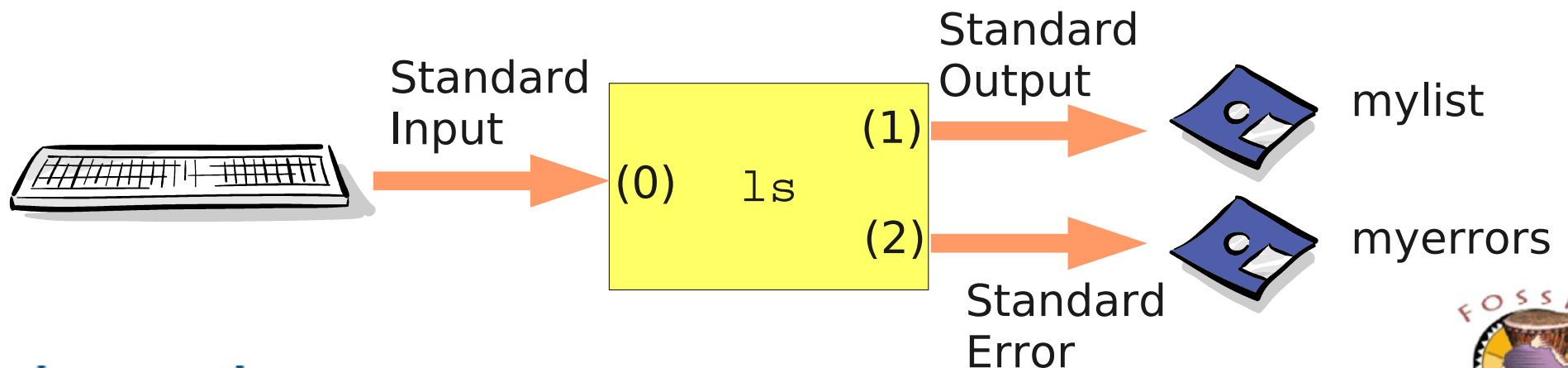
The output is *appended* to the file



Redirecting standard error

- Error messages are written to the *standard error* stream
 - The notation `2>` redirects standard error

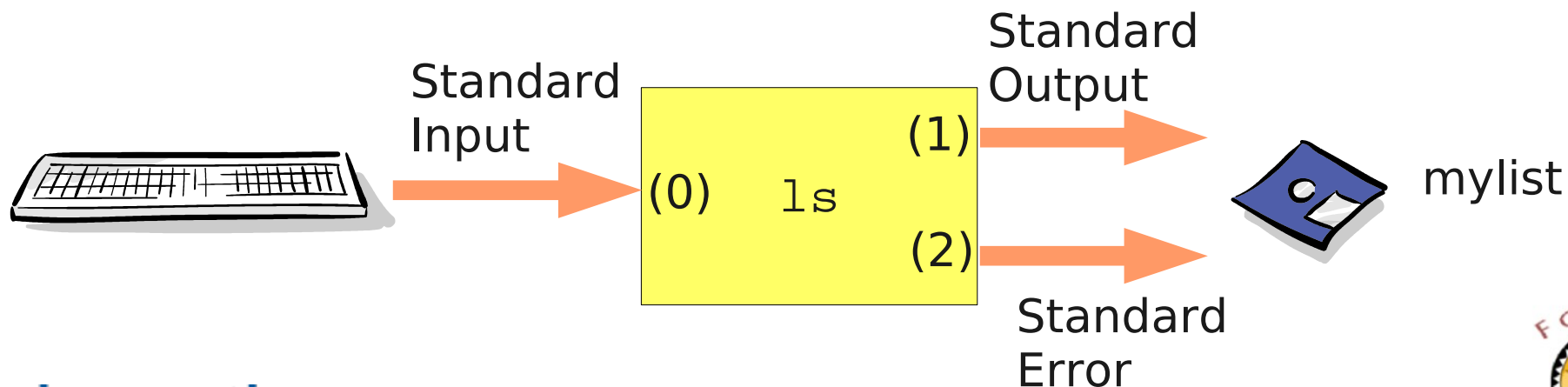
```
$ ls /opt /test > mylist
/bin/ls: /test: No such file or directory
$ ls /opt /test > mylist 2> myerrors
$ cat myerrors
/bin/ls: /test: No such file or directory
```



Combining standard error with standard output

- The notation '2>&1' says 'send output stream 2 (standard error) to wherever output stream 1 (standard output) is going'

```
$ ls /opt /test > mylist 2>&1
$ cat mylist
/bin/ls: /test: No such file or directory
/opt:
gnome
kde3
mozilla
```



Filter programs

- Programs such as `grep`, `wc`, `head`, `tail` and `sort` read their standard input if they are not given a filename argument
- Programs that read standard input, process it, and write the result to standard output are called *filters*

```
$ sort  
apple  
orange  
banana  
^D  
apple  
banana  
orange
```



Since no file name is given, `sort` reads from standard input (the keyboard)



The user enters `^D` to signify the end of input



The sorted output is written to standard output

Redirecting standard input

- The notation '<' redirects a program's standard input
- This example shows another filter, `tr`, performing lower to upper case conversion

```
$ tr a-z A-Z
```

```
Hello World
```

```
HELLO WORLD
```



Here, standard input comes from the keyboard; used ^D to terminate

```
$ tr a-z A-Z < /etc/motd
```



Standard input comes from a file

```
WELCOME TO FIRST TECHNOLOGY TRANSFER
```

- Standard input and standard output can both be redirected

```
$ tr a-z A-Z < /etc/motd > outfile
```

```
$ cat outfile
```

```
WELCOME TO FIRST TECHNOLOGY TRANSFER
```

Using programs in combination

- The output of one program may be used as input to another
 - An intermediate temporary file is one way to do this

```
$ grep DIY shopping > temp
```

```
$ sort -n -k 4 < temp
```

```
DIY          50   Nails          0.95
DIY           2   Screwdriver    2.00
DIY           2   Doorknob       8.40
DIY          20   Sandpaper     10.00
DIY           1   Hosepipe      15.00
```

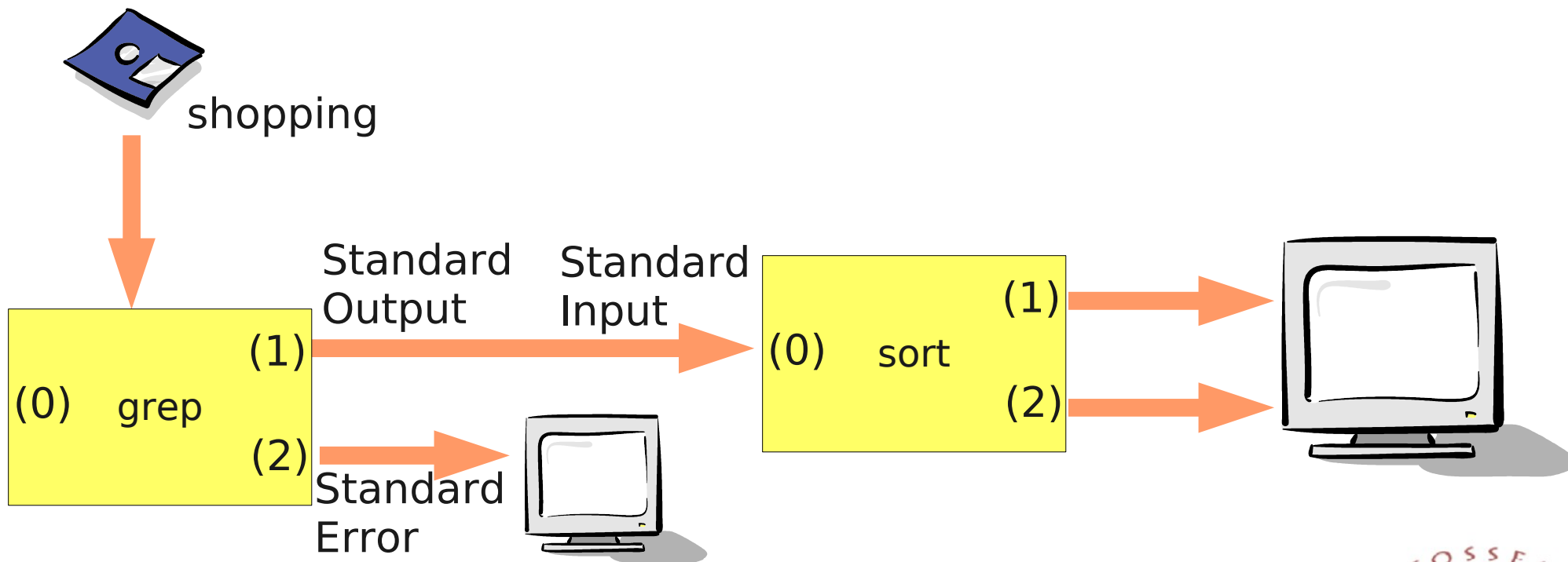
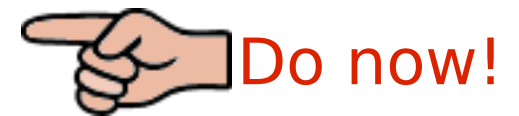
← Get list of all DIY items

← List sorted on price

Pipelines

- A pipe connects the standard output of one program directly to the standard input of another
 - The two programs are run concurrently

```
$ grep DIY shopping | sort -n -k 4
```



More pipeline examples

- How many items in the shopping list are from the Baker's?

```
$ grep Bakers shopping | wc -l
```

- Do a long listing of /etc, browse the output through less

```
$ ls -l /etc | less
```

- Find the most expensive item in the shopping list

```
$ sort -n -r -k 4 shopping | head -1
```

- How many files in /lib are actually directories?

```
$ ls -l /lib | grep '^d' | wc -l
```

Exercise: Using filters

- Display the first 10 lines of the file `/etc/sysconfig/network/config`
 - Hint: use filename completion to reduce typing
- Display the first 20 lines of the same file
 - Hint: use command history to reduce typing
- Show the last line (only) of the same file
- Search `/etc/sysconfig/network/config` for the string FIREWALL
 - Hint: use `grep`
- Redo the search, but ignore upper/lower case distinctions - i.e. search for `firewall`, `FIREWALL`, etc
- (Harder) Display all lines in `/etc/sysconfig/network/config` that are not comments (i.e. do not start with a '#')

Exercise: I/O Redirection and pipelines

- Create a file called `stuff1` containing a long listing of the files in `/etc`
 - Hint: Redirect standard output
- Search `stuff1` for lines containing the string `"rw-----"`, putting the output in a file called `stuff2`
- Count the number of lines in `stuff2`
 - Don't count them yourself, make the computer do it!
- Create a pipeline (no intermediate files) that displays a count of the number of files in `/etc` that have access mode `"rw-----"`
- Create a file called `stuff3` which contains, in order:
 - The current date and time
 - A calendar for the current year
 - The hostname of your machine, converted to upper case
 - Hint: append standard output to the file

Exercise: I/O Redirection and pipelines (contd)

- **Bonus Exercises:**
- Using a pipeline, display the line that describes the most expensive DIY item in the shopping list file
- Run the command `ps aux` and examine the output. It gives a list of all processes running on the computer. You're not expected to understand all of this output, but note that the first field shows the owner and the fifth field shows how much memory the process is using
- Using pipelines, devise commands to answer the following:
 - 1. How many processes are there altogether?
 - 2. How many processes are owned by root?
 - 3. How many processes are *not* owned by root?
 - 4. Which of root's processes is using the most memory? (Your pipeline should just display the line describing this process)

Managing files

- Managing files

— The current directory

— Creating and deleting directories

— Listing directory contents

— Copying files

— Renaming files

— Deleting files

— Updating files

— Links and the inode table

— Creating links to a file

— File and directory permissions revisited

Setting and querying the current directory

- Every process (including the shell) has a *current directory*
 - Where it will look for relative path names
 - The `cd` command changes the current directory.
 - The `pwd` command displays the name of the current directory

```
$ cd
$ pwd
/home/tux
$ cd pics
$ pwd
/home/tux/pics
$ cd /etc
$ pwd
/etc
$ cd -
/home/tux/pics
$ cd ..
$ pwd
/home/tux
```

`cd` with no arguments takes you to your home directory

Descend into a subdirectory

Change directory using an absolute pathname

'-' takes you back to your previous directory

'..' takes you up one level

Creating and deleting directories

- The `mkdir` command creates new directories
 - `-p` option creates additional directories, if required, along the path
- The `rmdir` command deletes directories
 - Only if they are empty

```
$ mkdir proposals/january
mkdir: cannot create directory `proposals/january':
  No such file or directory
$ mkdir -p proposals/january
$ rmdir proposals
rmdir: `proposals': Directory not empty
$ rmdir proposals/january
$ rmdir proposals
$
```

Listing directory contents with the `ls` command

- The `ls` command lists files in a directory
 - If a directory name is given, the *contents* of the directory are shown
 - With no arguments, the current directory is listed
- `ls` has many options. Here are a few:

Option	Meaning
None	Display filenames only, in a multi-column listing
<code>-l</code>	Display a 'long' listing, including file type, permissions, modification time, and size, in addition to the name
<code>-a</code>	Display 'hidden' files (ones whose names begin with a '.')
<code>-F</code>	After each name, append a character to indicate the file type: '/' indicates a directory, '*' indicates an executable file, '@' indicates a symbolic link
<code>-i</code>	Display inode numbers (discussed later)
<code>-t</code>	Sort by time of last alteration (by default, sort is alphabetic on file name)
<code>-u</code>	Sort by time of last access
<code>-R</code>	Recursive: descend into any subdirectories
<code>-d</code>	When listing a directory, list just the directory entry, not the contents

Hidden files

- Files whose names begin with a '.' are “hidden”
 - They do not show up on a normal directory listing
 - Most of them are configuration and startup files, for example:
 - `.bash_history`: Where the bash shell stores its command history
 - `.xinitrc`: Startup file for the X window system
 - `.bashrc`: Startup file for the bash shell
 - `.profile`: Startup file for all shells
- Use `ls -a` to show hidden files

Copying files with the `cp` command

- The `cp` command copies files

```
$ cp file1 file2
```

- This form makes a copy of `file1` under the name `file2`

```
$ cp file1 file2 ... dir
```

- This form makes copies of `file1 file2 ...` in (existing) directory `dir`

- Beware: the destination files will be replaced if they already exist

- Options include:

Option	Meaning
<code>-i</code>	Interactive mode: ask for confirmation before replacing an existing file
<code>-u</code>	If the destination file already exists, perform the operation only if the source file is newer than the destination file
<code>-l</code>	Create links instead of making copies
<code>-s</code>	Create symbolic links instead of making copies
<code>-r, -R</code>	Copy directories recursively

Renaming files with the `mv` command

- The `mv` command renames or moves files

```
$ mv file1 file2
```

This form renames `file1` as `file2`.

```
$ mv file1 file2 ... dir
```

This form moves `file1 file2 ...` into (existing) directory `dir`

- Beware: the destination files will be replaced if they already exist
- Options include:

Option	Meaning
<code>-i</code>	Interactive mode: ask for confirmation before replacing an existing file
<code>-u</code>	If the destination file already exists, perform the operation only if the source file is newer than the destination file

Deleting files with the `rm` command

- The `rm` command deletes files

```
$ rm file1 file2 ...
```

- Beware: there is no 'undelete' command!
- Options include:

Option	Meaning
-i	Interactive mode: ask for confirmation before deleting
-f	Normally <code>rm</code> will prompt for confirmation before deleting a file on which you do not have write permission. The <code>-f</code> flag suppresses this prompt and forces <code>rm</code> to delete the file
-r	Recursive mode: delete all subdirectories and contents

Updating files with the `touch` command

- The command `touch` updates the access and modification timestamps on a file to the current time
 - Makes it appear that the file has just been modified
 - If the file does not exist it is created with zero length

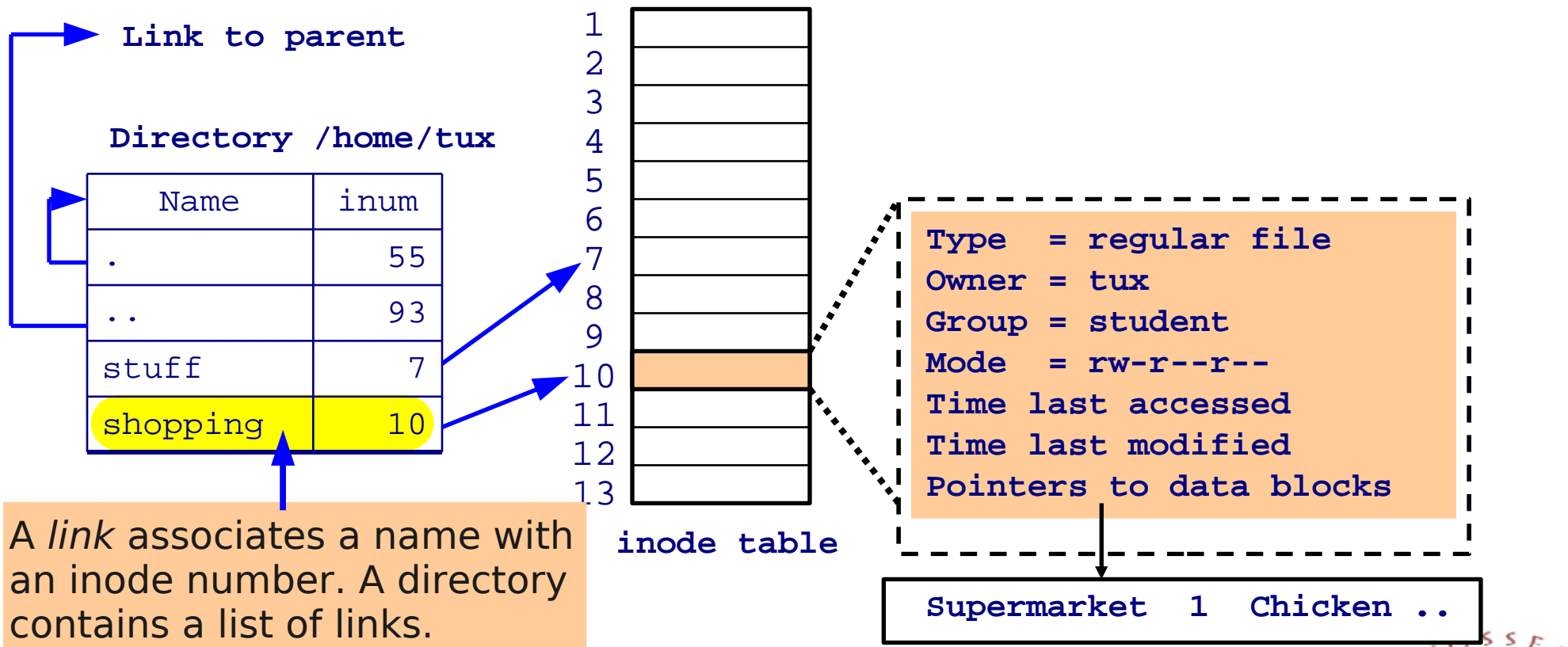
```
$ touch file1 file 2 ...
```

- Options for `touch` include:

Option	Meaning
<code>-a</code>	Update only the access timestamp
<code>-m</code>	Update only the modification timestamp
<code>-r file</code>	Update the timestamps to match those of file, not the current time
<code>-t time</code>	Update the timestamps to the specified time, in the format <code>[[CC]YY]MMDDhhmm[.ss]</code>

Links and the inode table

- The filesystem associates a structure called an *inode* with each file
 - Contains file's attributes and pointers to the actual data blocks
 - space for inodes (inode table) is pre-allocated when filesystem created



Creating links to a file with `ln`

- Creating additional links to a file allows the file to be referenced by more than one name
- General form of `ln` command is:

```
$ ln existing_name new_name
```

- Example:

```
$ cd /home/dilbert
```

```
$ ln ../tux/shopping mylist
```

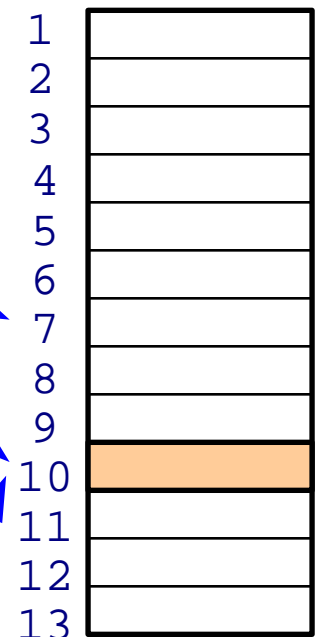
The system keeps count (in the inode) of the number of links. When the last link is removed, the inode and the data blocks are freed.

Directory /home/tux

Name	inum
.	55
..	93
foo	7
shopping	10

Directory /home/dilbert

Name	inum
.	1236
..	93
bar	12
mylist	10



inode table

Familiarising with links

- The `-i` option to `ls` shows the inode numbers

```
$ ls -li
total 4
 190116 -rw-r--r--  1 tux  users  491  2004-04-19  11:22  shopping
$ ln shopping mylist
$ ls -li
total 8
 190116 -rw-r--r--  2 tux  users  491  2004-04-19  11:22  mylist
 190116 -rw-r--r--  2 tux  users  491  2004-04-19  11:22  shopping
$ rm shopping
$ ls -li
total 4
 190116 -rw-r--r--  1 tux  users  491  2004-04-19  11:22  mylist
```



The inode number



The link count

Symbolic links

- A *symbolic link* is a special type of file that simply contains the pathname of a “target” file
 - Any references to the symbolic link are automatically translated into references to the target
- The '-s' flag tells `ln` to create symbolic links
- ```
$ ln -s shopping mylist
$ ls -l shopping mylist
lrwxrwxrwx 1 chris users 8 2004-04-23 11:43 mylist -> shopping
-rw-r--r-- 1 chris users 491 2004-04-23 09:24 shopping
```
- Deleting the target file breaks the symbolic link and causes confusing behaviour:

```
$ rm shopping
$ less mylist
mylist: No such file or directory
```

# File and directory permissions revisited

- The ways in which file access permissions control what you can do derive from a few simple rules
  - To access the data in a file, you need read permission on the file
  - To modify the data in a file, you need write permission on the file
  - To list the names (only) of the files in a directory, you need read permission on the directory
  - To list the attributes of the files in a directory (e.g. to perform an `ls -l`), or to use the directory in a pathname, or to make it your current directory, you need execute permission on the directory
  - (You need both read and execute permission to have useful access to a directory)
  - To add a link to a directory, or to remove a link from a directory, you need write permission on the directory

# File permission examples

| Operation                   | Situation            | Required permissions                                                   |
|-----------------------------|----------------------|------------------------------------------------------------------------|
| <code>cp file1 file2</code> | file2 doesn't exist  | r permission on file1<br>w permission on file2's directory             |
| <code>cp file1 file2</code> | file2 already exists | r permission on file1<br>w permission on file2                         |
| <code>mv file1 file2</code> |                      | w permission on file1's directory<br>w permission on file2's directory |
| <code>rm file1</code>       |                      | w permission on file1's directory                                      |
| <code>ln file1 file2</code> |                      | rx permission on file2's directory; w permission on file2's directory  |

# File permission quiz

- Using the `ls` command, and your knowledge of file permissions, predict which of the following operations would be allowed if you were logged in as `tux`

| Operation                                                            | Allowed? | Why or why not? |
|----------------------------------------------------------------------|----------|-----------------|
| Make a copy of <code>/etc/at.deny</code> in your home directory      |          |                 |
| Make a copy of <code>/etc/group</code> in your home directory        |          |                 |
| Make a copy of your shopping file in the <code>/etc</code> directory |          |                 |
| Delete your own home directory                                       |          |                 |

# Exercise: Managing files

1. Log in as `tux`
2. Copy the files `passwd`, `group`, `hosts` and `fstab` from `/etc` to your home directory
3. In your home directory, rename the file `fstab` to `table`
4. Create a directory called `private` in your home directory
5. Change the permissions on `private` so that only you can access it
  - Hint: the permissions should be `rwX-----`
6. Move the files `passwd` and `group` into `private`
7. Change directory into the `private` directory and list the files there
8. Log out and log back in as `dilbert` with password `adams`
  - Can you list `tux`'s home directory? \_\_\_\_\_
  - Can you list `tux`'s `private` directory? \_\_\_\_\_

# Exercise: Managing files (continued)

9. Log out as `dilbert` and log back in as `tux`
10. Create a hard link called `mygroup` in your home directory to the file `group` in the `private` directory
  - What is the inode number associated with this link? \_\_\_\_\_
11. Create a symbolic link called `mypasswd` in your home directory to the file `passwd` in the `private` directory
12. Delete the `private` directory and its contents
13. Try to access the files `mygroup` and `mypasswd` in your home directory
  - What happens? \_\_\_\_\_
  - Can you explain why? \_\_\_\_\_
14. Delete the symbolic link `mypasswd`

**End of Exercise**

# Miscellaneous features

- Miscellaneous features

File name expansion using wildcards

Editing with vi

Finding files with find

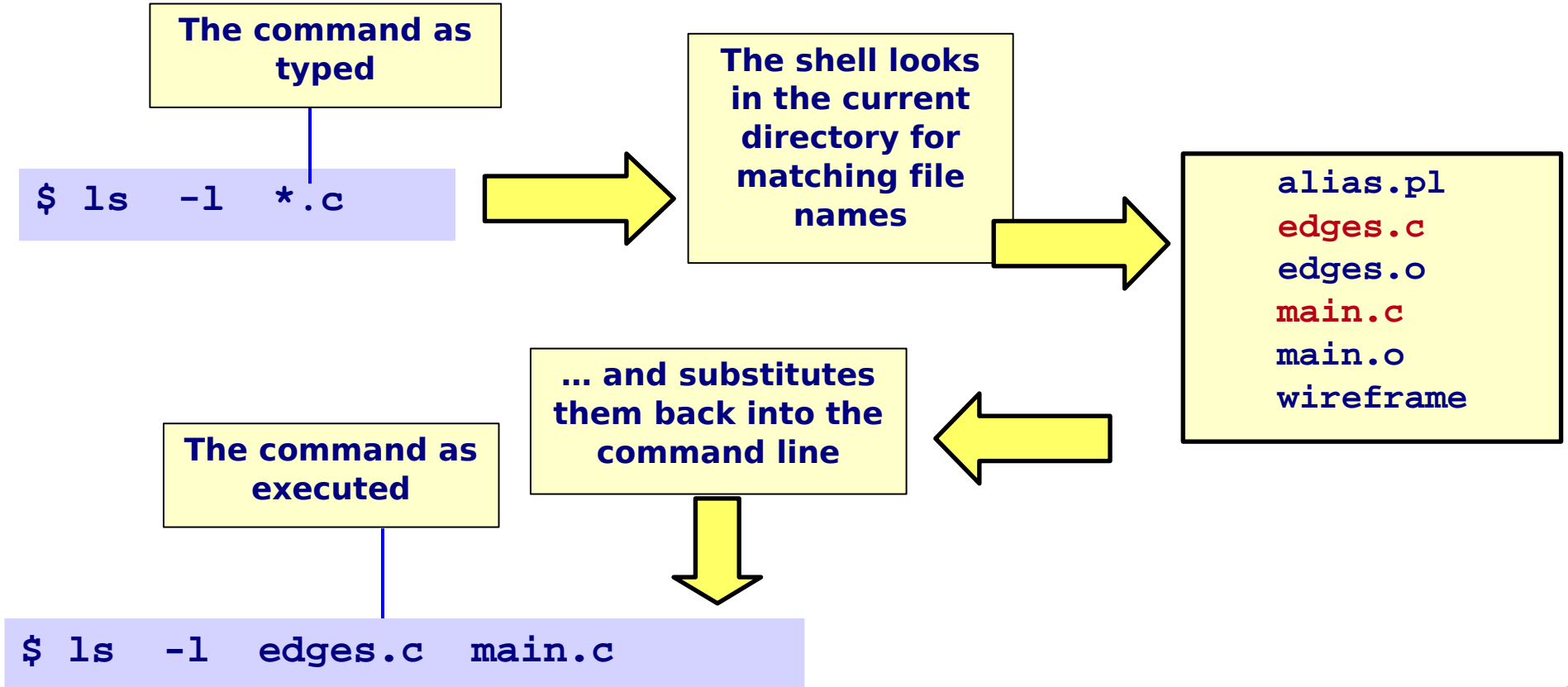
Manual pages

Builtin help

HOWTO documents

# Filename expansion using wildcards

- The shell uses several meta-characters for matching patterns in file names
  - Commonly known as *wildcards*



# Filename expansion using wildcards (continued)

- \* Matches zero or more characters
- ? Matches exactly one character
- [ ] Matches any one of the enclosed characters, e.g. [AaTt]
- [x-y] Matches any character in range  
e.g. [a-m] [A-Z] [0-9]
- Linux has no notion of filename “extensions”
  - \* matches all names, like \*.\* in DOS
- Multiple wildcards can be used
  - E.g. **rm [A-Z]\*.html**
- Wildcards can be used in multiple components of a pathname
  - E.g. **rm backups/\*.199[7-9]/expenses??**

# Wildcard quiz

Given these files in the current directory



```
410-chap1.doc intro.old
410-chap2.doc meetings.June
410-chap3.doc meetings.July
410-chap4.doc meetings.Aug
410-chap5.doc oldstuff
410-CHAPS.doc opensource
display openwindows
display.c project6
display.h project45
display.object project46
ideas project346
ideas.old training
ideas.older venues
index windows
intro x-windows
```

A directory

What do these commands do?



```
rm *.old
ls -l 410-chap?.doc
less 410-chap[2-5].doc
mv ideas* training
mv ideas.* training
rm *old*
rm *
cp [v-z]* training
cp [a-z]* training
ls display.?
rm proj*6
rm proj*[a-z]6
```

# Exercise: Wildcards

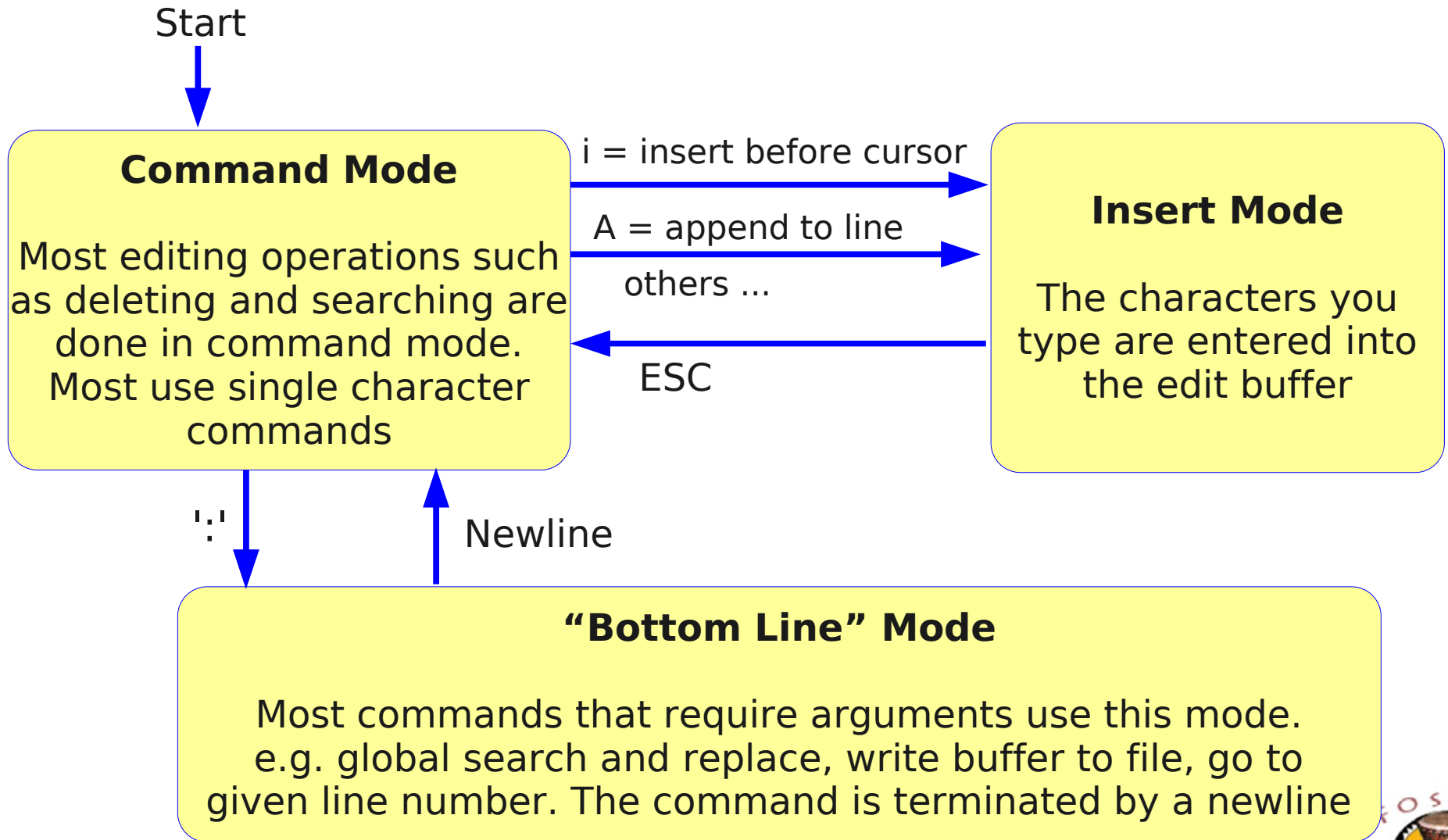
- Do a long listing of all the files in `/usr/bin` whose names contain a digit
  - How many such files are there?
- Create a subdirectory called `config` in your home directory
- Copy all the files whose name ends in `.conf` from `/etc` to your `config` directory
  - Can you find a way to suppress the error messages due to unreadable files?
- Change the access permissions of all files in your `config` directory to be `rw-----`
- Are there any files in your `config` directory whose name begins with a vowel?
- Delete any files in your `config` directory whose name begins with a vowel

# Editing with `vi`

- The editor `vi` is the 'standard' editor on UNIX and Linux systems
- Advantages:
  - It is available on every UNIX and Linux system
  - It is available on the SuSE rescue disk
  - It works on character terminals, without a graphical user interface
  - It is fast and powerful once you know it well
- Disadvantages:
  - It is a 'moded' editor which makes it difficult to learn initially
  - There are a lot of commands to remember
- Professional linux users and administrators benefit in the long term by learning `vi`
  - Budget several hours of learning time, over a period of time

# Working modes in vi

- vi has three major operating modes



# Command mode in vi

- In command mode, most 'ordinary' characters invoke a command
  - This table shows a minimal command set to survive with vi. It is nowhere near complete and does not show the full power of the command set.

| Command  | Meaning                                                    |
|----------|------------------------------------------------------------|
| i        | Switch to insert mode, insert text before cursor position  |
| A        | Switch to insert mode, append to current line              |
| x        | Delete the character under the cursor                      |
| dd       | Delete the current line (and put in the paste buffer)      |
| D        | Delete to end of line                                      |
| /pattern | Search for pattern, forwards from current cursor position  |
| ?pattern | Search for pattern, backwards from current cursor position |
| n        | Repeat the search in the same direction                    |
| N        | Repeat the search in the opposite direction                |
| yy       | 'Yank' (copy) the current line into the paste buffer       |
| p        | Insert the paste buffer before the current cursor position |
| ZZ       | Save the file and exit (this is the normal way to exit vi) |
| .        | Repeat the last change at the current cursor position      |

# “Bottom line” mode in vi

- The table below shows some important 'bottom line' commands
  - There are also powerful global 'search and replace' commands that are not shown here

| Command | Meaning                                                                        |
|---------|--------------------------------------------------------------------------------|
| :q      | Quit the editor (only works if no changes have been made)                      |
| :q!     | Quit the editor, abandoning any changes that have been made                    |
| :wq     | Write the file out and exit (same as 'ZZ' in command mode)                     |
| :w file | Write the edit buffer out to the specified file (instead of the original file) |
| :41     | Go to line 41                                                                  |

# Exercise: Using vi

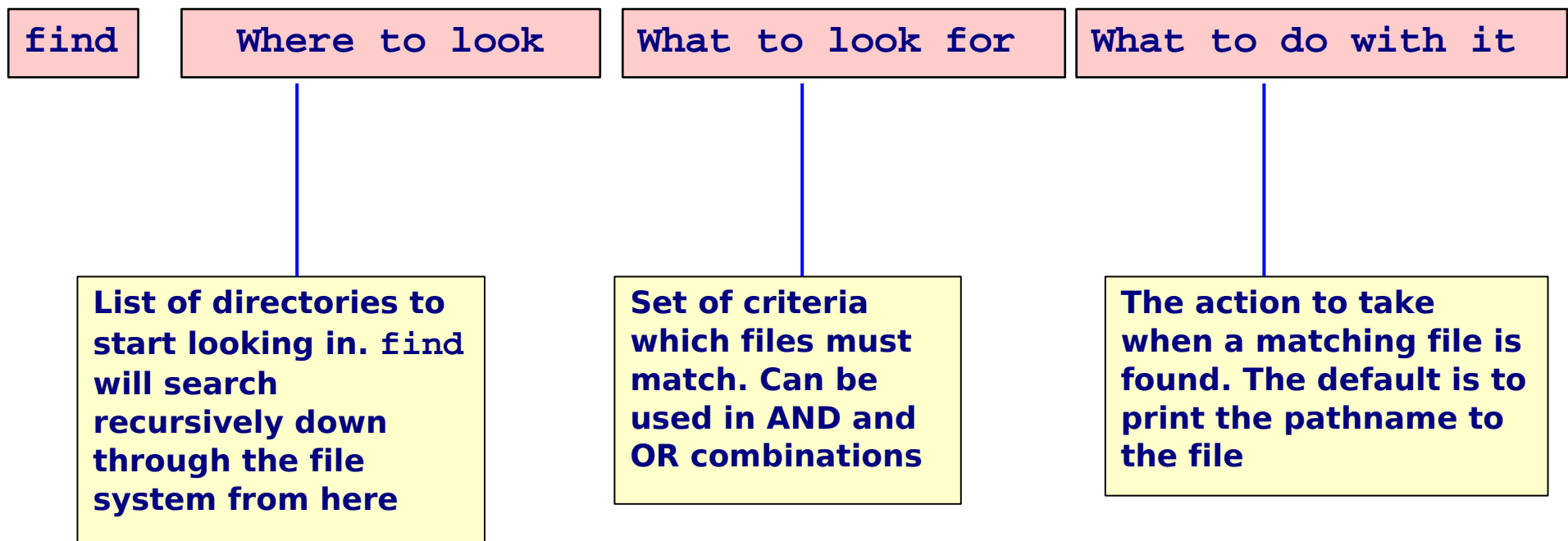
- Using vi, enter the text:

```
A linux sysadmin called Pete
Typed all his commands with his feet
After "rm -r", his toe hit a star,
A sysadmin no longer is Pete
```

- Save the text to the file `limerick` in your home directory
- Make the following changes:
  - Change 'Pete' to 'Joe', (twice) and 'feet' to 'toe'
  - Swap the first line with the last line. (Use cut-and-paste, don't retype!)
  - Delete the third line
- Save the result in the file `junk` (*not* to the `limerick` file)

# Finding files with the `find` command

- The `find` command searches for files meeting specified criteria
  - Name, owner, group, timestamps, permissions, size, etc.
- `find` has a complicated syntax; the general framework is:



# Search criteria for `find`

- Search criteria for `find` include:

| Syntax                        | Description                                                  | Example                     |
|-------------------------------|--------------------------------------------------------------|-----------------------------|
| <code>-name 'string'</code>   | File name matches 'string'. Wildcards are allowed            | <code>-name '*.old'</code>  |
| <code>-iname 'string'</code>  | Same as <code>-name</code> but not case sensitive            | <code>-name 'greet*'</code> |
| <code>-user username</code>   | File is owned by username                                    | <code>-user dilbert</code>  |
| <code>-group groupname</code> | File belongs to groupname                                    | <code>-group root</code>    |
| <code>-type d, f, or l</code> | File is a directory, regular file, or symbolic link          | <code>-type f</code>        |
| <code>-size +N</code>         | File is bigger than N blocks<br>suffix c = bytes, k = kbytes | <code>-size +1000k</code>   |
| <code>-size -N</code>         | File is smaller than N blocks                                | <code>-size -50c</code>     |

- The default action for `find` is simply to display the names of the matching files

# Some examples of using `find`

- Show all files ending in '.c' in (and below) the current directory

```
$ find . -name '*.c'
```

- Find all files under `/home` owned by `tux`

```
$ find /home -user tux
```

- Find ordinary files in `/usr/bin` which are bigger than 1 Mbyte

```
$ find /usr/bin -type f -size +1000k
```

- Find all ordinary files owned by `root` which have zero length

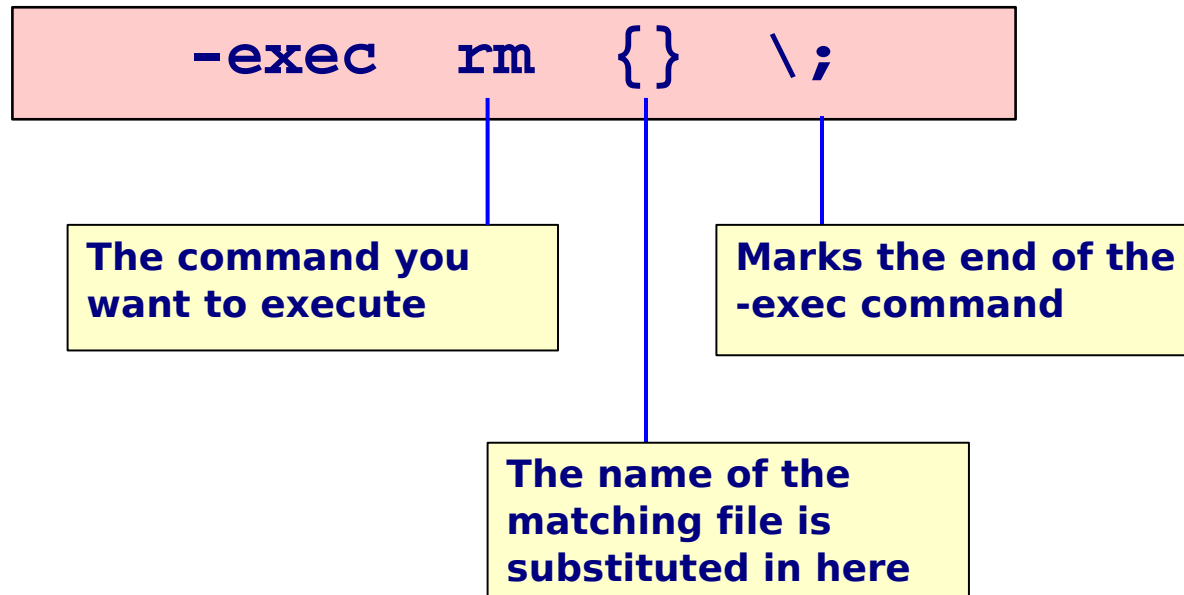
```
$ find / -type f -user root -size 0 2> /dev/null
```

# More search criteria for `find`

| Syntax                  | Description                                     | Example                 |
|-------------------------|-------------------------------------------------|-------------------------|
| <code>-perm xxx</code>  | File permissions exactly match octal digits xxx | <code>-perm 644</code>  |
| <code>-perm +xxx</code> | File has at least one of the permissions xxx    | <code>-perm +222</code> |
| <code>-perm -xxx</code> | File has all of the permissions xxx             | <code>-perm -001</code> |
| <code>-mtime +n</code>  | File last modified more than n days ago         | <code>-mtime +14</code> |
| <code>-mtime -n</code>  | File last modified less than n days ago         | <code>-mtime -2</code>  |
| <code>-atime +n</code>  | File last accessed more than n days ago         | <code>-atime +7</code>  |
| <code>-atime -n</code>  | File last accessed less than n days ago         | <code>-atime -1</code>  |

# Actions for `find`

- An action specifies what to do with each matching file
  - `print` (also the default) just writes out the pathname of the file
  - `ls` writes output similar to `ls -li` for the file
- Any arbitrary command may be executed using `-exec`
  - The syntax is messy; here's an example:



# More examples of using `find`

- Find all files in `/home/tux` or `/home/dilbert` which are world-writable and give a detailed listing

```
$ find /home/tux /home/dilbert -perm +002 -ls
```

- Delete files under `/home` with names ending  `'.bak'` which have not been accessed for two weeks

```
$ find /home -name '*.bak' -atime +14 -exec rm {} \;
```

- Find all files which are not symbolic links but have mode `777`

```
$ find / ! -type l -perm 777 2> /dev/null
```

↑  
'!' negates the sense of the following test

↑  
Discard reports of unreadable directories, etc.

# Exercise: Using `find`

- List all the directories under `/home` that belong to `dilbert`
- How many symbolic links are there under `/usr/bin`?
- Does `root` own any zero-length regular files? How many?
  - Hint: don't count them yourself!
- What is the largest file in the filesystem?
  - Hint: it is bigger than 10 Mbytes

# Manual pages

- The traditional way of providing online help is the “manual page”
  - accessed via the `man` command

```
tux@earth:~> man mount
```

```
MOUNT(8) Linux Programmer's Manual MOUNT(8)
```

## NAME

```
mount - mount a file system
```

## SYNOPSIS

```
mount [-lhV]
```

```
mount -a [-fFnrsvw] [-t vfstype] [-O optlist]
```

```
mount [-fnrsvw] [-o options [,...]] device | dir
```

```
mount [-fnrsvw] [-t vfstype] [-o options] device dir
```

## DESCRIPTION

All files accessible in a Unix system are arranged in one big tree, the file hierarchy, rooted at `/`. These files can be spread out over several devices. The `mount` command serves to attach the file system found on some device to the big file tree. Conversely, the `umount(8)` command will detach it again.



# How a manual page is organised

- Each manual page is divided up into a number of sections
  - Not all sections are present for all commands

| Section          | Contents                                                          |
|------------------|-------------------------------------------------------------------|
| NAME             | Name and short description of the command                         |
| SYNOPSIS         | Description of the syntax                                         |
| DESCRIPTION      | Detailed description of the command                               |
| OPTIONS          | Description of all available options                              |
| COMMANDS         | Instructions that can be given to the program while it is running |
| FILES            | Files referenced by the command                                   |
| SEE ALSO         | List of related commands                                          |
| DIAGNOSTICS      | Possible error messages and what they mean                        |
| EXAMPLE          | Examples of usage                                                 |
| AUTHOR           | Who wrote it                                                      |
| BUGS or WARNINGS | Known errors and problems                                         |

# Manual page section numbering

- Each manual page is allocated a 'section number' depending on what kind of thing it describes:

| Section | Contents                             | Used by                     |
|---------|--------------------------------------|-----------------------------|
| 1       | Commands for end users               | End users                   |
| 2       | System Calls                         | Developers                  |
| 3       | Functions and Library Routines       | Developers                  |
| 4       | Device Files                         | Administrators / developers |
| 5       | Configuration files and file formats | Administrators              |
| 6       | Games                                | ?                           |
| 7       | Macro packages and file formats      | ?                           |
| 8       | System Administration Commands       | Administrators              |

- The output of `man` is piped through `less` for ease of browsing

# Manual page section numbering (continued)

- By default the man command finds the requested page in the lowest-numbered section
  - If the same name appears in more than one section, you may have to specify the section number explicitly to get the right man page

```
$ man crontab
```

```
... man page for crontab command in section 1 ...
```

```
$ man 5 crontab
```

```
... man page for crontab file format in section 5 ...
```

```
$ man uname
```

```
... man page for uname command in section 1 ...
```

```
$ man 2 uname
```

```
... man page for uname system call in section 2 ...
```



# Searching the manual pages by keyword

- You can search for manual pages by keyword
  - `man -k keyword` or `apropos keyword`
  - keyword must appear in NAME section of man page

```
$ apropos partition
mpartition (1) - partition an MSDOS hard disk
sfdisk (8) - Partition table manipulator for Linux
gpart (8) - guess PC-type hard disk partitions
ntfsfix (8) - tool for fixing NTFS partitions altered by ...
mkfs.jfs (8) - create a JFS formatted partition
lvmdiskscan (8) - scan for all disks / multiple devices ...
jfs_mkfs (8) - create a JFS formatted partition
pvcreate (8) - initialize a disk or partition for use by LVM
cfdisk (8) - Curses based disk partition table manipulator
partprobe (8) - inform the OS of partition table changes
fdisk (8) - Partition table manipulator for Linux
parted (8) - a partition manipulation program
```

# Exercises: Using the manual pages

1. For this exercise, log in as `tux`
2. Look up the man page for the `ls` command
  - What option gives a 'long' listing? \_\_\_\_\_
  - What option gives a color listing? \_\_\_\_\_
3. Look up the man page for the `write` command
  - What command did you use? \_\_\_\_\_
4. Look up the man page for the `write` system call
  - What command did you use? \_\_\_\_\_
5. Find the name of the command for formatting floppy disks
  - Hint: Use `apropos` (and maybe `grep`)
6. Find the name of (a) the command and (b) the system call for changing permissions on a file

# Built-in Help

- Most of the GNU command line tools have built-in help, using the `--help` option

```
tux@earth:~> cat --help
Usage: cat [OPTION] [FILE]...
Concatenate FILE(s), or standard input, to standard output.

-A, --show-all equivalent to -vET
-b, --number-nonblank number nonblank output lines
-e equivalent to -vE
-E, --show-ends display $ at end of each line
-n, --number number all output lines
-s, --squeeze-blank never more than one single blank line
-t equivalent to -vT
-T, --show-tabs display TAB characters as ^I
-u (ignored)
-v, --show-nonprinting use ^ and M- notation, except for LFD and TAB
--help display this help and exit
--version output version information and exit
```

With no FILE, or when FILE is -, read standard input.



## 3.3 HOWTOs

- A huge number of 'HOWTO' documents provide help on a very wide variety of topics
  - From Astronomy to ZIP drives
- The HTML versions are available in the package `howtoenh`
  - In the directory `/usr/share/doc/howto/en/html`
  - May not be installed by default
- Be aware that HOWTOs are maintained by a very loosely knit group of people on a best effort basis. They may be
  - Out of date, poorly written, or wrong
  - Intended for a Linux distribution other than SuSE
- Online repository of HOWTOs at [www.linuxdoc.org](http://www.linuxdoc.org)

## 3.4 SuSE Support Database and Hardware Database

- SuSE maintain an online support database at:
  - <http://sdb.suse.de/sdb/en/html/index.html>
- There is also a hardware compatibility database at:
  - <http://cdb.suse.de>
- Other useful web sites include:
  - <http://www.linux.org>
  - <http://www.linux.com>
  - <http://en.tldp.org>
  - <http://www.linuxplanet.com>
  - <http://www.cert.org> (Security related)
  - <http://www.kernel.org> (Kernel related)

# End of chapter quiz

- What command would you use to:

Browse through a text file

Display the first 5 lines of a file

List a directory

Display the name of your current directory

Count the number of lines in a file

Create a symbolic link

Create a directory

Copy a file

Delete a file

Find a file based on its size and owner

Look up on-line documentation

Display your user ID

Search a file for a string

Delete an (empty) directory

Delete a non-empty directory