



The Filesystem

Dr. Gareth Roy (x6439)
gareth.roy@glasgow.ac.uk

- The Filesystem
- Navigating the filesystem
- Users & Permissions
- Working with files
- Archives

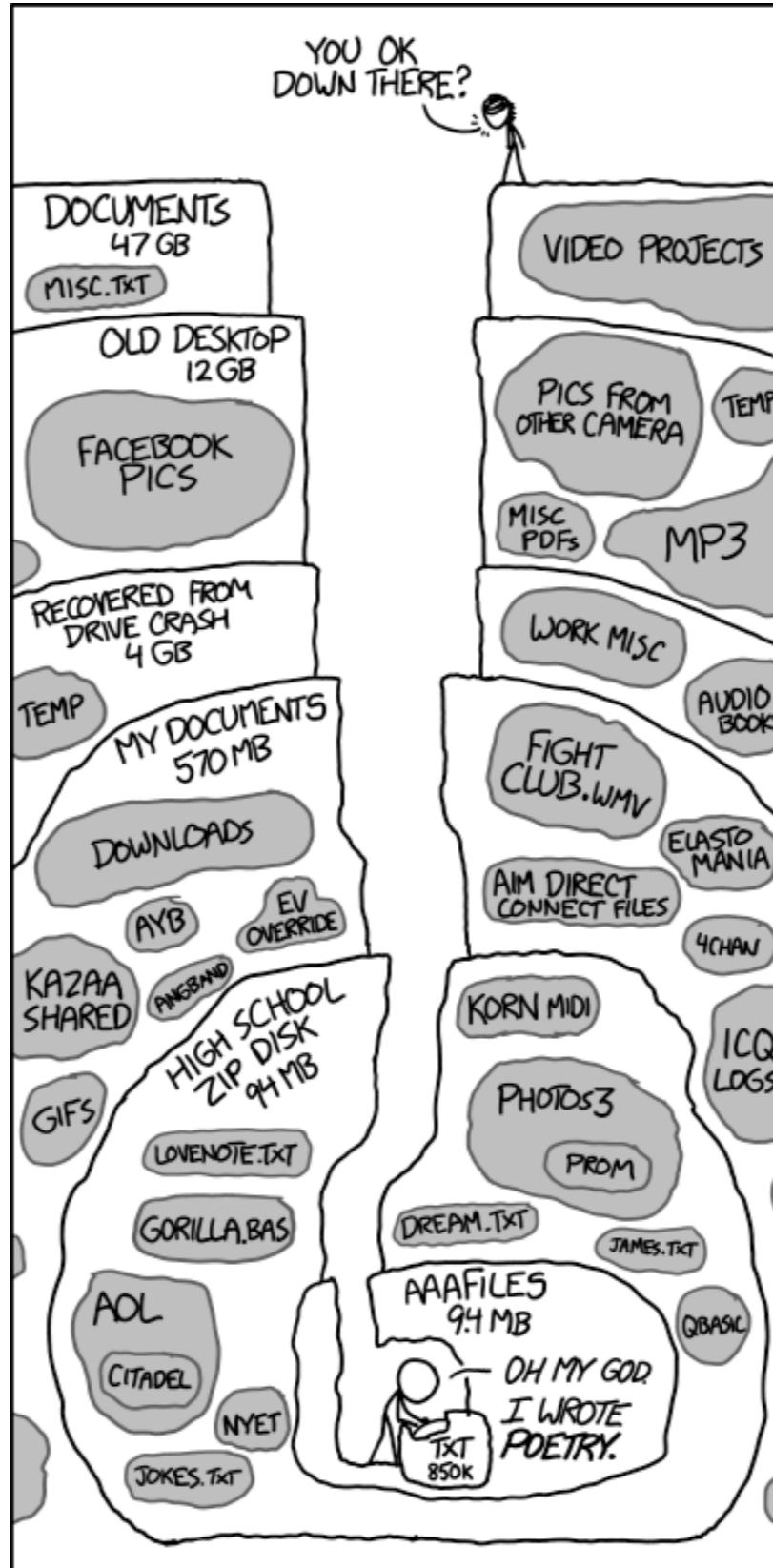
- The Filesystem
- Navigating the filesystem
- Users & Permissions
- Working with files
- Archives

"I think the major good idea in Unix was its clean and simple interface: open, close, read, and write."

— Ken Thompson

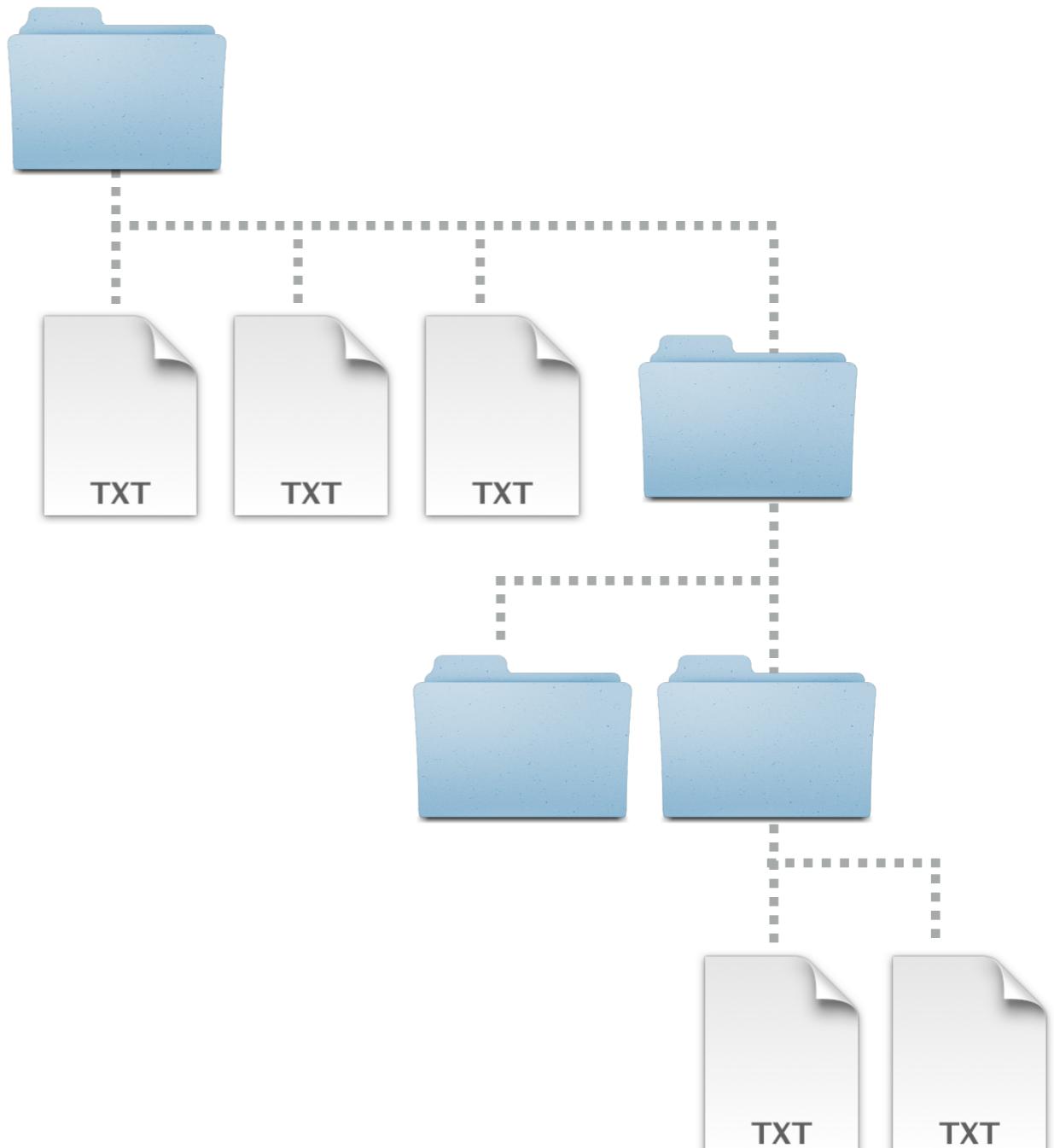
"On a UNIX system, everything is a file; if something is not a file, it is a process."

— Unknown



The Filesystem

- The Linux filesystem is hierarchical.
- It has a tree like structure that begins at the “root” node.
- Each node in the tree can be a file or directory (or link, socket...).
- A directory is a special file that contains a list of other files, and can include other directories.
- Each “file” in the filesystem has a name, which is case sensitive (afile is not the same as AFILE).
- Each “file” or node in the tree has a unique identifier called it's **inode**.

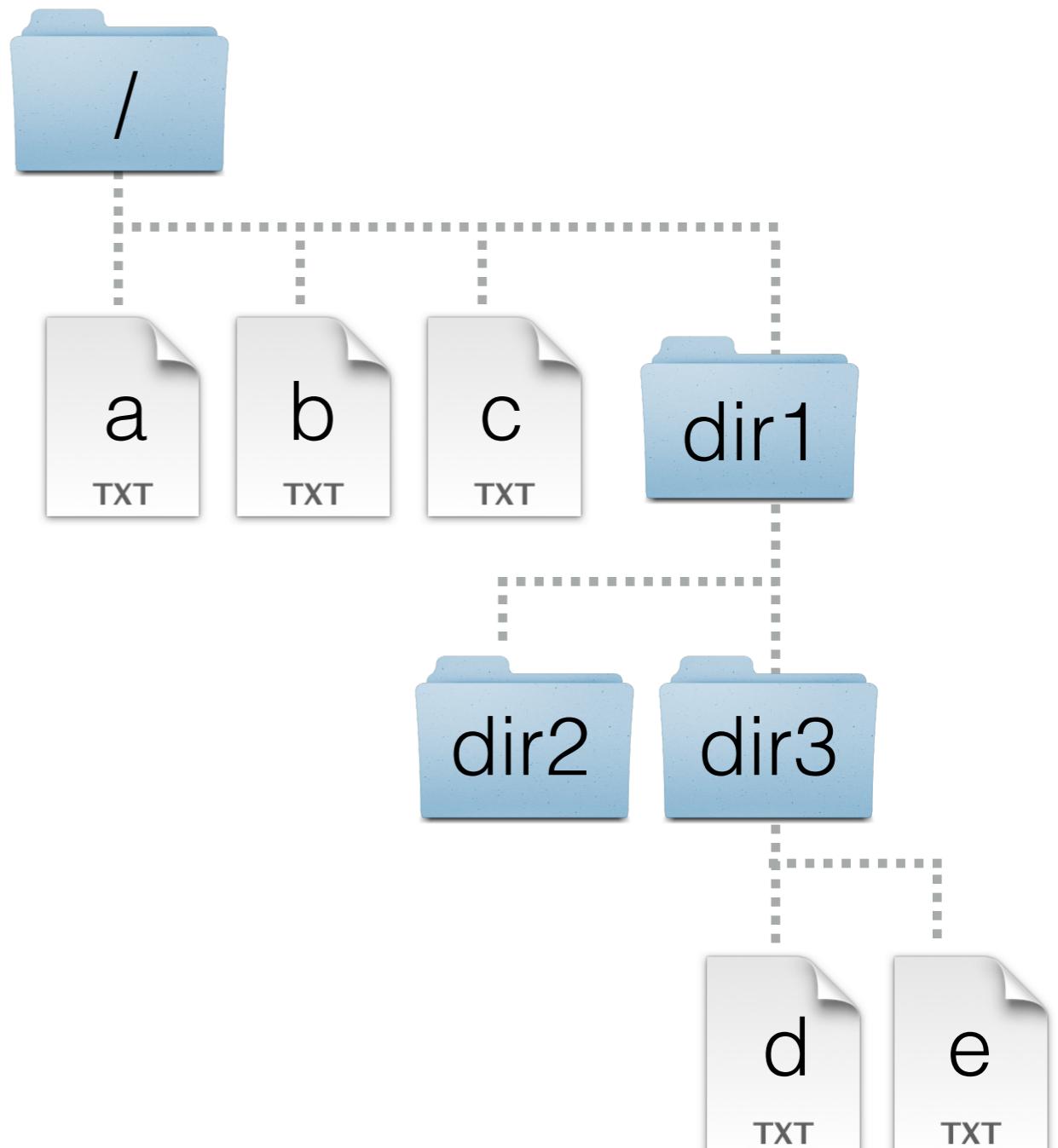


File types

Type	Symbol	Purpose
Regular	-	Standard text files
Directories	d	Files that are lists of files
Links	l	A special file that allows another file to appear in multiple locations in the filesystem.
Special File	c	Input / Ouput sources (usually located in /dev)
Sockets	s	inter process networking similar to TCP/IP
Named Pipes	p	Similar to a socket, allows process to communicate with one another.

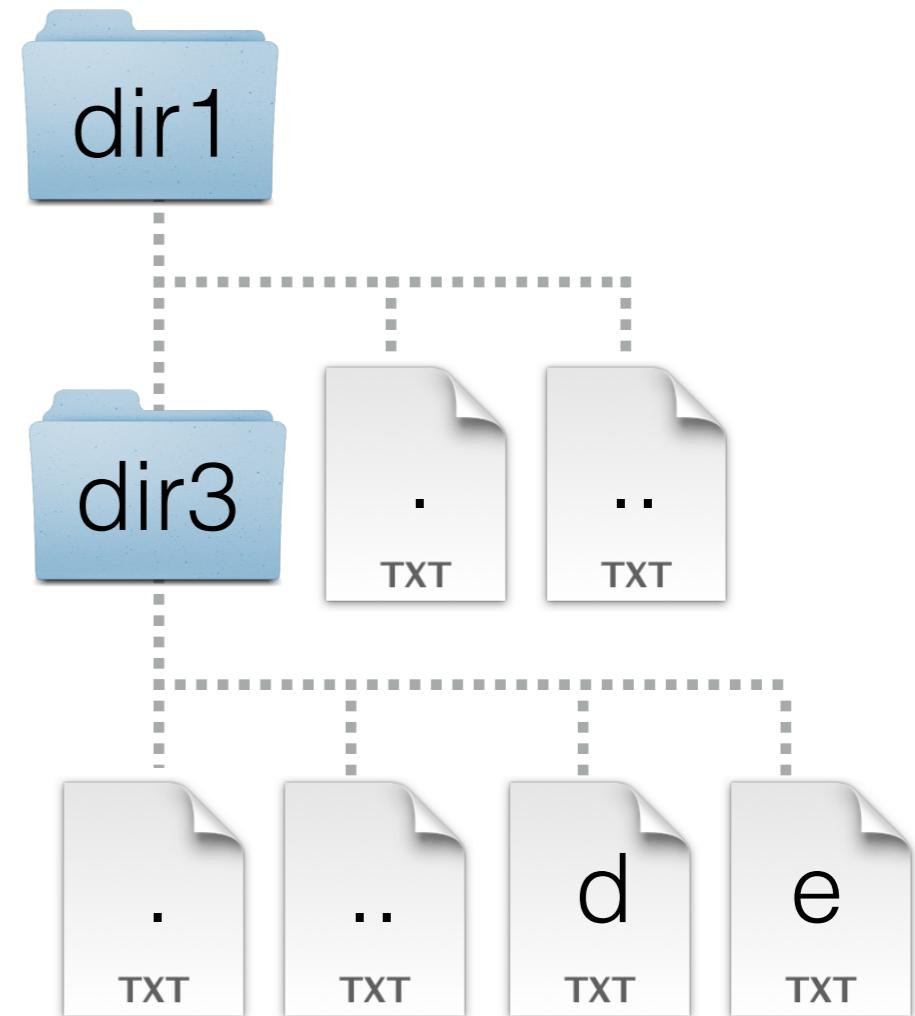
Absolute File Paths

- The location of each file is given by its **path**.
- The absolute path starts from the “root” node denoted by a **/**
- In this example the absolute path for the file ‘a’ would be **/a**
- Each time we descend into a directory we add an additional **/**
- So the absolute path for the file ‘e’ would be **/dir1/dir3/e**
- We can locate any file in the filesystem by using it’s absolute path.



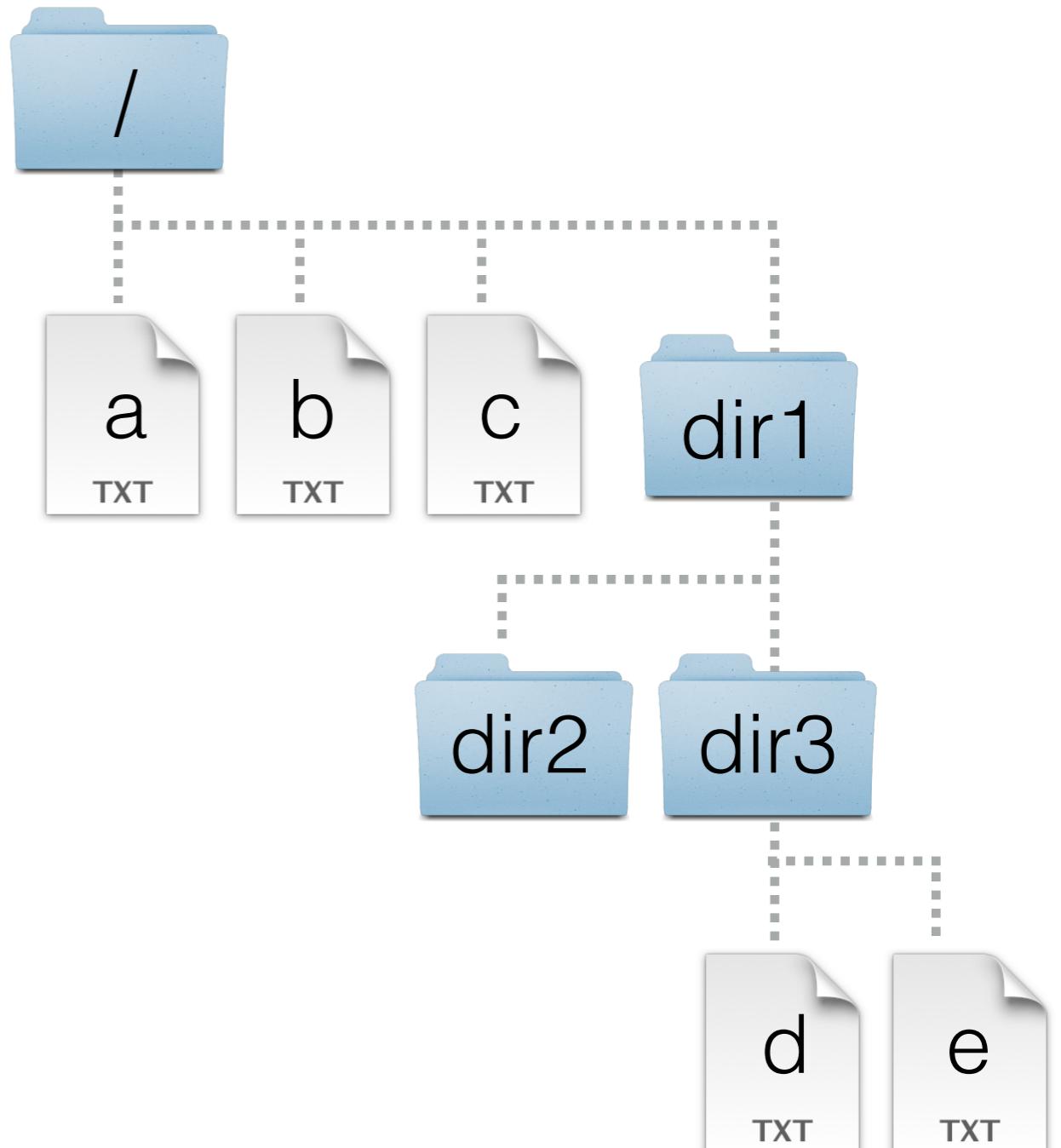
Hidden Files

- In Linux files can be hidden so they don't appear in a directory.
- Hidden files in Linux all begin with '.' character
- This is often used for configuration files, like **.vimrc**
- There are two special hidden files found in every directory.
- **.** - the current directory.
- **..** - the directory above this one in the hierarchy.



Relative File Paths

- Relative paths can be used to references files at other parts of the file system.
- This is useful as absolute paths can make a system inflexible or difficult to modify.
- For instance if we were presently working in **dir2** we could access file '**e**' via: **../dir3/e**
- This can be interpreted as a set of actions:
 - go up one level (..)
 - enter **dir3**
 - access file '**e**'



Standard Filesystem Locations

- **/etc** - holds a lot of configuration files, e.g. /etc/cups controls the configuration of the printer queues and which devices are made available at boot.
- **/dev** - holds links to the various pieces of hardware being managed by the kernel
- **/proc** - holds views into the current kernel operating parameters and processes
- **/var** - holds ‘variable’ data, e.g. logging files, mail, printer spools
- **/usr** - holds binaries, documentation, software libraries
- **/home** - is the collection of private directories for users of the system
- **/bin** - contains the binaries that are essential to the system (shells, cp, mv, rm, cat, ls, etc...)
- **/sbin** - contains the binaries that are essential to the system but only for use by administrators

- The Filesystem
- Navigating the filesystem
- Users & Permissions
- Working with files
- Archives

Looking Around

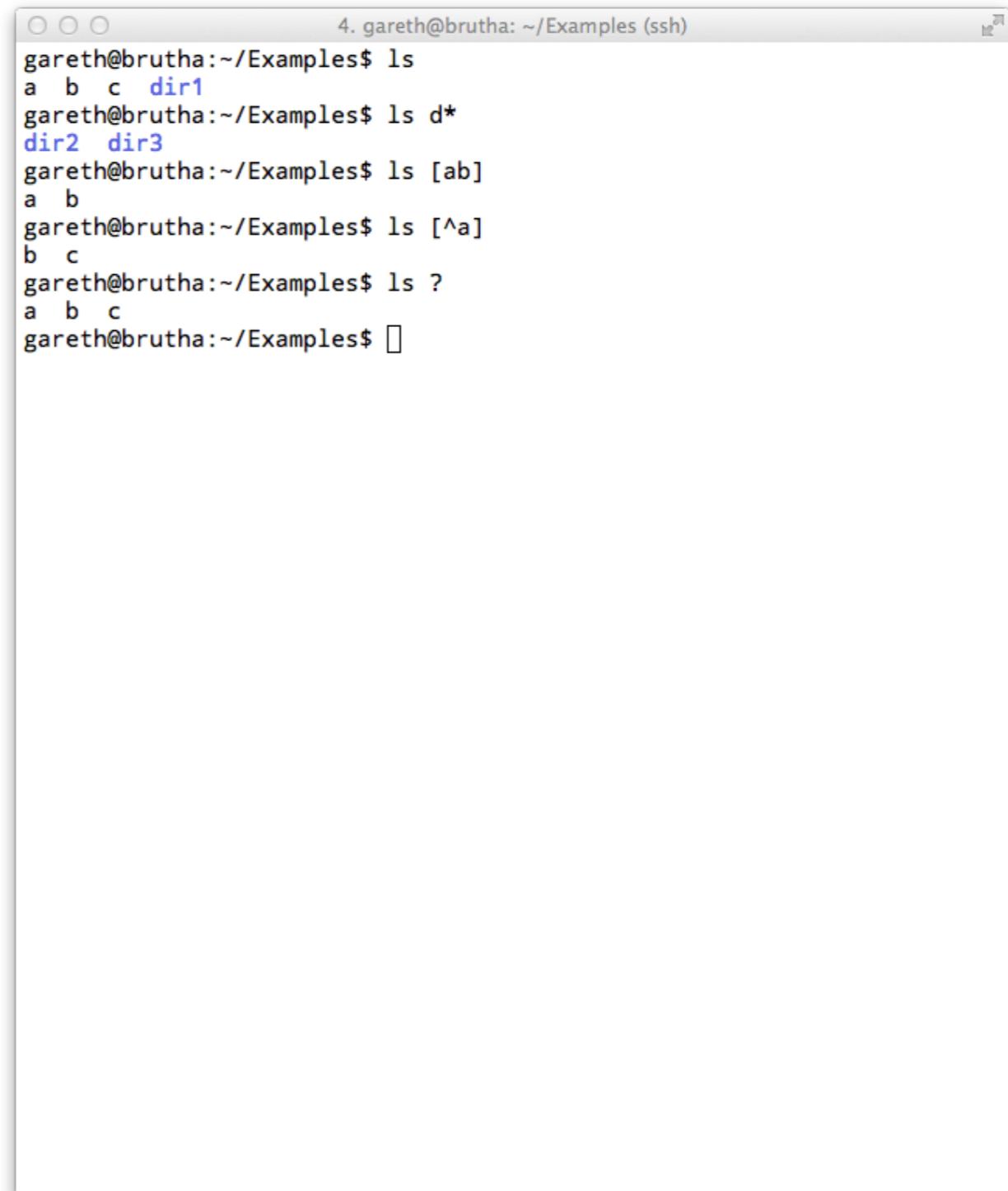
- **pwd** - tells us our current position in the filesystem (print working directory).
- **tree** - prints out the complete ‘tree’ of our current location, good for getting a visual representation of a directory.
- **ls** - lists the contents of a directory. By itself defaults to the current directory, but can be given an absolute or relative path.
- **ls -a** - the minus a option tells ls to show hidden files.
- **ls -l** - the minus l option tells ls print the contents of a directory in long format.

```
4. gareth@brutha: ~/Examples (ssh)
gareth@brutha:~/Examples$ pwd
/home/gareth/Examples
gareth@brutha:~/Examples$ gareth@brutha:~/Examples$ gareth@brutha:~/Examples$ tree
.
├── a
└── b
    ├── c
    └── dir1
        ├── dir2
        └── dir3
            ├── d
            └── e

3 directories, 5 files
gareth@brutha:~/Examples$ gareth@brutha:~/Examples$ gareth@brutha:~/Examples$ ls
a b c dir1
gareth@brutha:~/Examples$ gareth@brutha:~/Examples$ gareth@brutha:~/Examples$ ls -a
. ... a b c dir1
gareth@brutha:~/Examples$ gareth@brutha:~/Examples$ gareth@brutha:~/Examples$ ls -l
total 4
-rw-rw-r-- 1 gareth gareth 0 Jan 14 09:05 a
-rw-rw-r-- 1 gareth gareth 0 Jan 14 09:05 b
-rw-rw-r-- 1 gareth gareth 0 Jan 14 09:05 c
drwxrwxr-x 4 gareth gareth 4096 Jan 14 09:28 dir1
gareth@brutha:~/Examples$ 
```

File Globbing

- **ls** can also be used to list specific files, and the list can be filtered by using wildcards.
- Wildcards are symbols that have special meanings when combined with filenames.
 - * - matches all characters
 - ? - matches only one character.
 - [] - specifies alphanumeric values to match on i.e. [a-c]
 - ^ - negates a match
- For example we could match all the files that have a 'txt' extension by the following:
 - **ls *.txt**
- Or we could match on files whose name are only a single character
 - **ls ?**

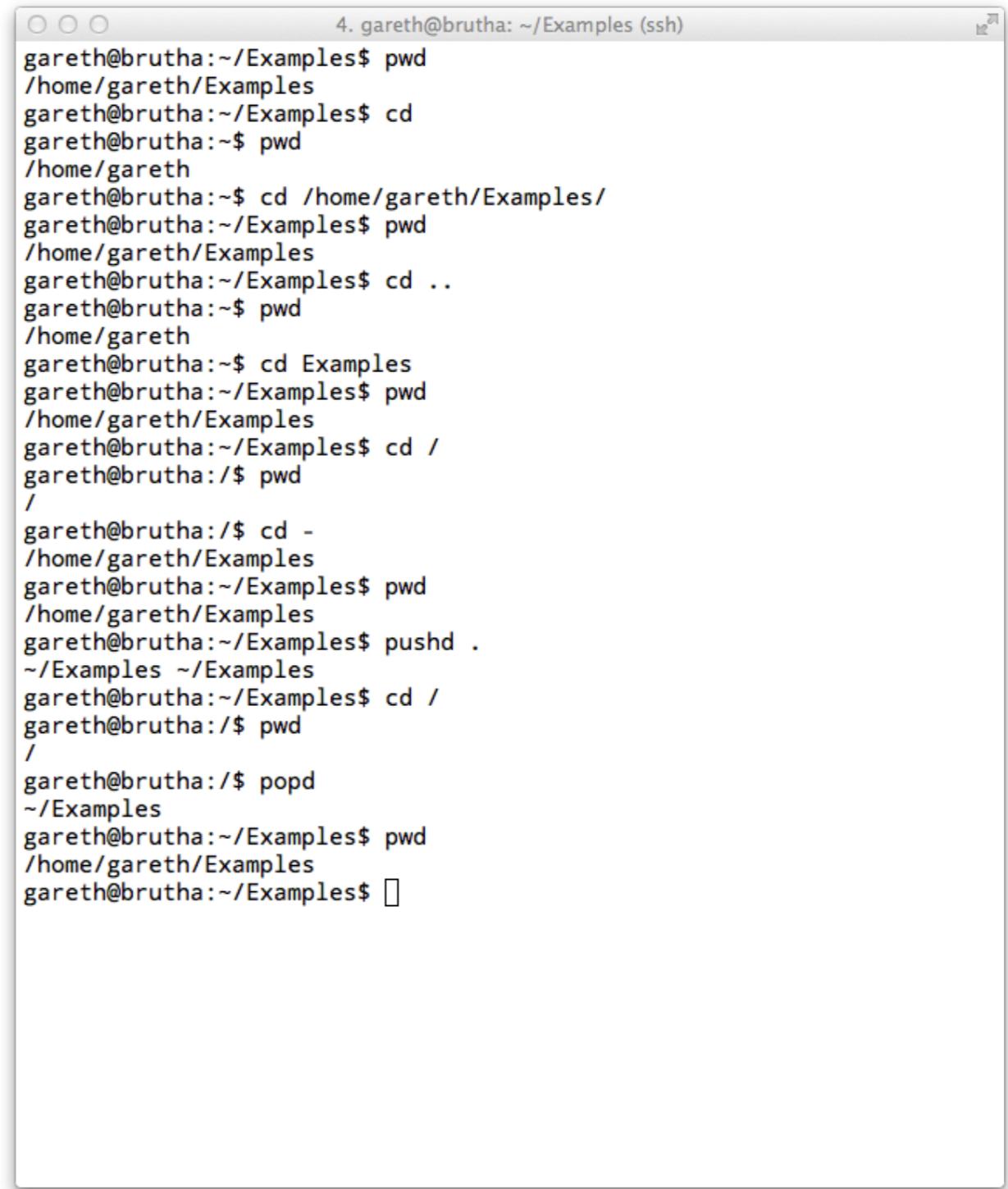


The screenshot shows a terminal window titled "4. gareth@brutha: ~/Examples (ssh)". It displays several commands and their outputs:

```
gareth@brutha:~/Examples$ ls
a b c dir1
gareth@brutha:~/Examples$ ls d*
dir2 dir3
gareth@brutha:~/Examples$ ls [ab]
a b
gareth@brutha:~/Examples$ ls [^a]
b c
gareth@brutha:~/Examples$ ls ?
a b c
gareth@brutha:~/Examples$ 
```

Moving Around

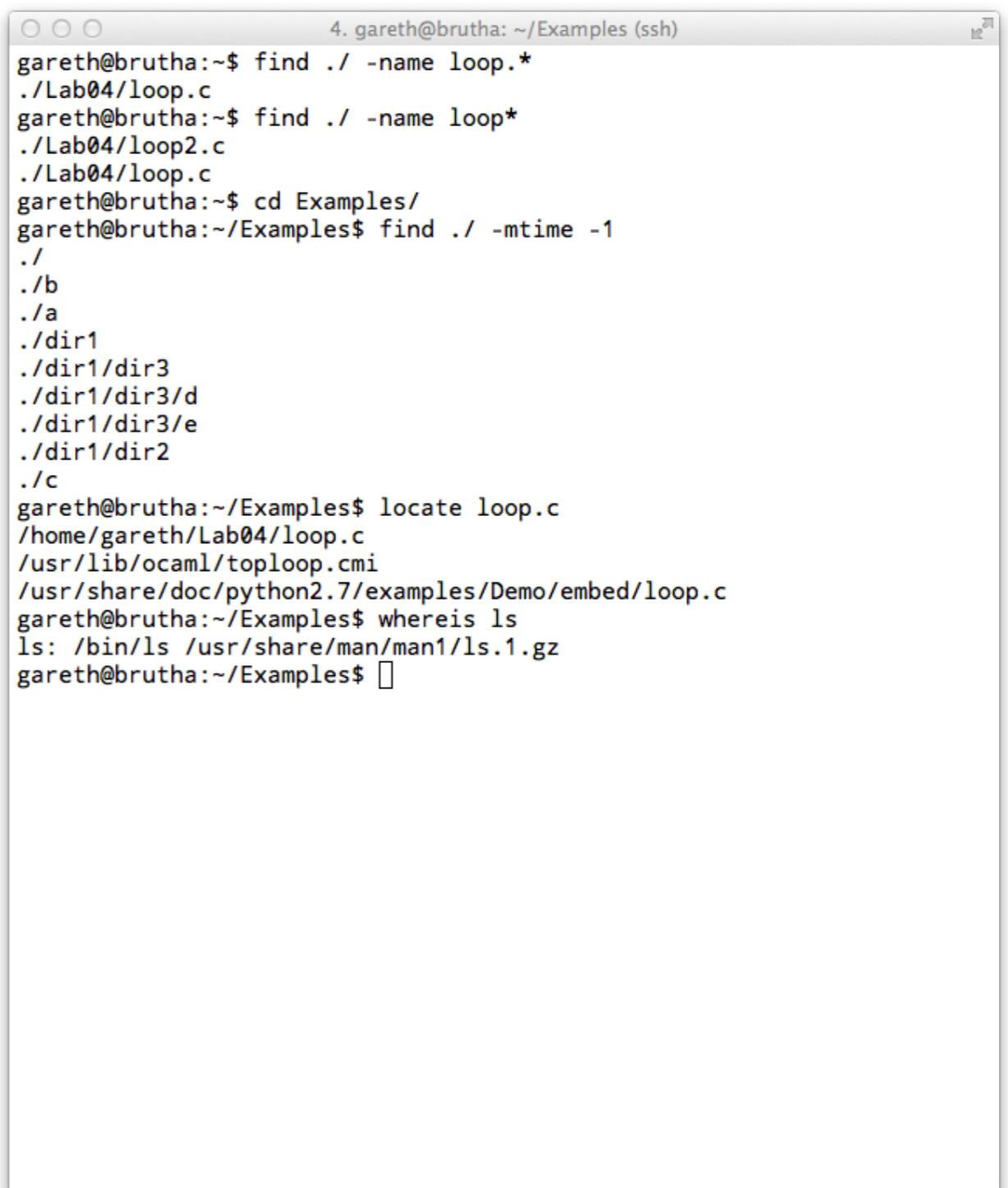
- **cd** - command to “change directory”
- **cd** - with no arguments will return you to your home directory.
- **cd <dir>** - changes you to a directory <dir>. This can be an absolute or relative path.
- **cd -** - will return you to the previous directory.
- **pushd** - stores a directory path you'd like to visit later
- **popd** - takes the first directory stored and changes to it.



```
4. gareth@brutha: ~/Examples (ssh)
gareth@brutha:~/Examples$ pwd
/home/gareth/Examples
gareth@brutha:~/Examples$ cd
gareth@brutha:~$ pwd
/home/gareth
gareth@brutha:~$ cd /home/gareth/Examples/
gareth@brutha:~/Examples$ pwd
/home/gareth/Examples
gareth@brutha:~/Examples$ cd ..
gareth@brutha:~$ pwd
/home/gareth
gareth@brutha:~$ cd Examples
gareth@brutha:~/Examples$ pwd
/home/gareth/Examples
gareth@brutha:~/Examples$ cd /
gareth@brutha:$ pwd
/
gareth@brutha:$ cd -
/home/gareth/Examples
gareth@brutha:~/Examples$ pwd
/home/gareth/Examples
gareth@brutha:~/Examples$ pushd .
~/Examples ~/Examples
gareth@brutha:~/Examples$ cd /
gareth@brutha:$ pwd
/
gareth@brutha:$ popd
~/Examples
gareth@brutha:~/Examples$ pwd
/home/gareth/Examples
gareth@brutha:~/Examples$
```

Finding Files

- **find** - used to search through the filesystem to find a file.
 - `find ./ -name Lab04`
 - `find ./ -mtime +2`
 - `find ./ -mtime -3`
- **locate** - accesses a database of files (usually built once a day), is faster than find but only as good as the database.
- Use find to get newer files, and if you want to filter on things like modification or creation time
- **whereis** - will return the location of programs and manpages
 - `whereis ls`



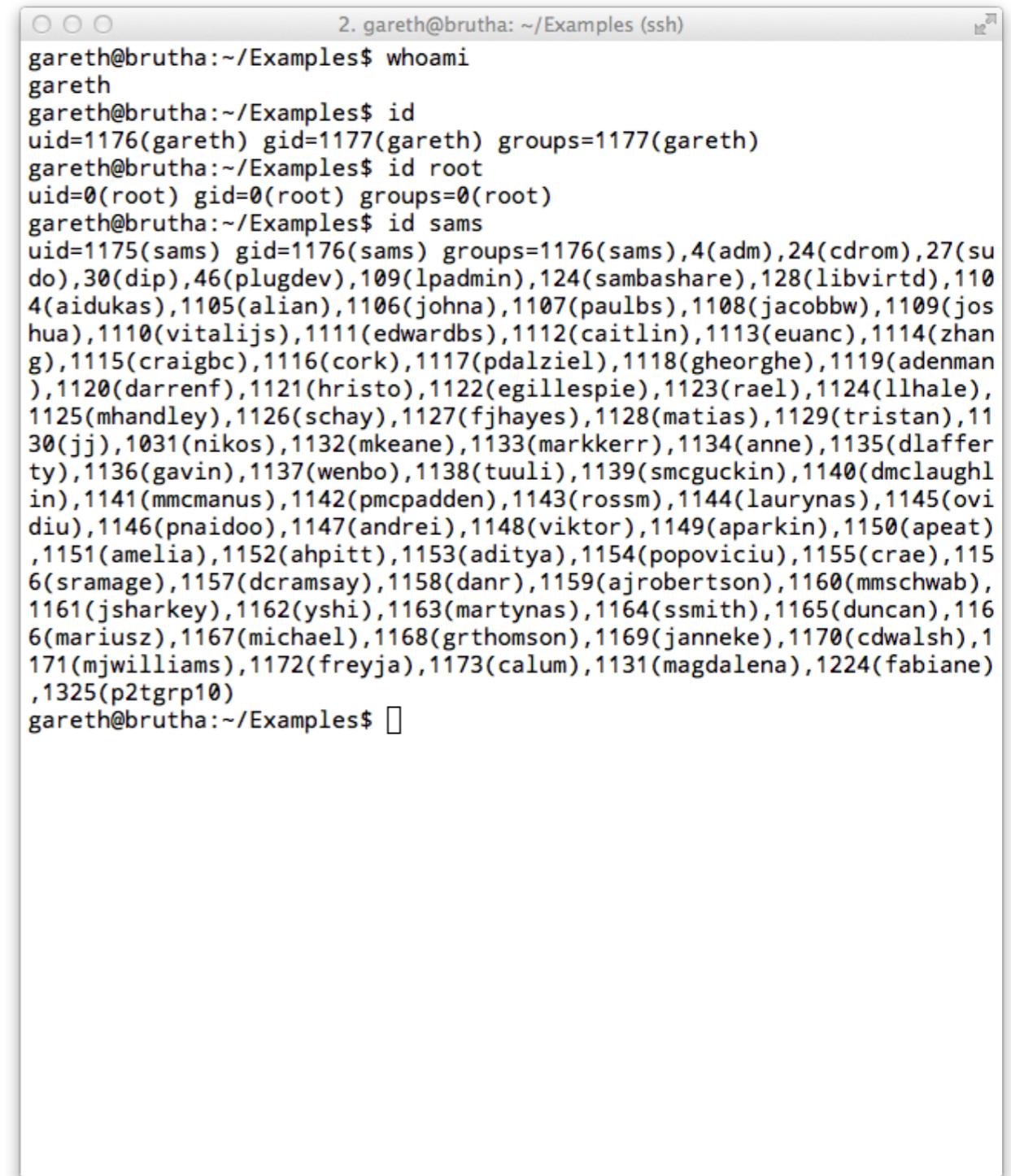
A screenshot of a terminal window titled "4. gareth@brutha: ~/Examples (ssh)". The window displays several command-line sessions demonstrating file search tools:

```
gareth@brutha:~/Examples$ find ./ -name loop.*  
./Lab04/loop.c  
gareth@brutha:~/Examples$ find ./ -name loop*  
./Lab04/loop2.c  
./Lab04/loop.c  
gareth@brutha:~/Examples$ cd Examples/  
gareth@brutha:~/Examples$ find ./ -mtime -1  
./  
.b  
.a  
.dir1  
.dir1/dir3  
.dir1/dir3/d  
.dir1/dir3/e  
.dir1/dir2  
.c  
gareth@brutha:~/Examples$ locate loop.c  
/home/gareth/Lab04/loop.c  
/usr/lib/ocaml/toploop.cmi  
/usr/share/doc/python2.7/examples/Demo/embed/loop.c  
gareth@brutha:~/Examples$ whereis ls  
ls: /bin/ls /usr/share/man/man1/ls.1.gz  
gareth@brutha:~/Examples$ 
```

- The Filesystem
- Navigating the filesystem
- Users & Permissions
- Working with files
- Archives

Users & Groups

- Linux is a multi-user operating system.
- Each User has a unique ID (**uid**) and is associated with a set of groups. Each group also has a unique id (**gid**)
- What a user can do is restricted by the rights associated with each user or group.
- **whoami** - tells you what your username is.
- **id** - tells you about your id and group info.
- Linux systems have the concept of a “root” (or super) user who can do anything on the system
- This right can also be delegated to a specific user, often referred to **sudoer** (super user do).



A screenshot of a terminal window titled "2. gareth@brutha: ~/Examples (ssh)". The window displays the output of several commands:

```
gareth@brutha:~/Examples$ whoami
gareth
gareth@brutha:~/Examples$ id
uid=1176(gareth) gid=1177(gareth) groups=1177(gareth)
gareth@brutha:~/Examples$ id root
uid=0(root) gid=0(root) groups=0(root)
gareth@brutha:~/Examples$ id sams
uid=1175(sams) gid=1176(sams) groups=1176(sams),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),109(lpadmin),124(sambashare),128(libvirtd),1104(aidukas),1105(alian),1106(johna),1107(paulbs),1108(jacobbw),1109(joshua),1110(vitalijs),1111(edwardbs),1112(caitlin),1113(euanc),1114(zhang),1115(craigbc),1116(cork),1117(pdalziel),1118(gheorghe),1119(adenman),1120(darrenf),1121(hristo),1122(egillespie),1123(rael),1124(ljhale),1125(mhandley),1126(schay),1127(fjhayes),1128(matias),1129(tristan),1130(jj),1031(nikos),1132(mkeane),1133(markkerr),1134(anne),1135(dlafferty),1136(gavin),1137(wenbo),1138(tuuli),1139(smcguckin),1140(dmclaughlin),1141(mmcmanus),1142(pmcpadden),1143(rossm),1144(laurynas),1145(ovidiu),1146(pnaidoo),1147(andrei),1148(viktor),1149(aparkin),1150(apeat),1151(amelia),1152(ahpitt),1153(aditya),1154(popoviciu),1155(crae),1156(sramage),1157(dcramsay),1158(danr),1159(ajrobertson),1160(mmschwab),1161(jsharkey),1162(yshi),1163(martynas),1164(ssmith),1165(duncan),1166(mariusz),1167(michael),1168(grthomson),1169(janneke),1170(cdwalsh),1171(mjwilliams),1172(freyja),1173(calum),1131(magdalena),1224(fabiane),1325(p2tgrp10)
gareth@brutha:~/Examples$
```

Permissions

- All restrictions in a Linux system are handled at the file level.
- Each file or directory has the notion of ownership by a user or by a group.
- Additionally restrictions are placed on what can be done to a file.
- ls -l will show the contents of a directory in long format. This shows:
 - permissions
 - no. of links
 - owner
 - group owner
 - size of file in bytes
 - date modified
 - name

```
gareth@brutha:~/Examples$ ls  
a b c d dir1  
gareth@brutha:~/Examples$ ls -ltr  
total 4  
-rw-rw-r-- 1 gareth gareth 0 Jan 14 09:05 a  
-rw-rw-r-- 1 gareth gareth 0 Jan 14 09:05 b  
drwxrwxr-x 4 gareth gareth 4096 Jan 14 09:28 dir1  
-rw-rw-r-- 1 gareth gareth 0 Jan 14 13:59 c  
lrwxrwxrwx 1 gareth gareth 11 Jan 15 08:56 d -> dir1/dir3/d  
gareth@brutha:~/Examples$
```

```
gareth@brutha:~/Examples$ ls -ltr  
total 4  
-rw-rw-r-- 1 gareth gareth 0 Jan 14 09:05 a  
-rw-rw-r-- 1 gareth gareth 0 Jan 14 09:05 b  
drwxrwxr-x 4 gareth gareth 4096 Jan 14 09:28 dir1  
-rw-rw-r-- 1 gareth gareth 0 Jan 14 13:59 c  
lrwxrwxrwx 1 gareth gareth 11 Jan 15 08:56 d -> dir1/dir3/d
```

Permissions

Filetype → -rwxrw-rw-

User Group Other

- First part is filetype
 - **d** - directory
 - **l** - link
 - **-** - normal file
- Next is permissions, split into three parts
 - **User** permissions
 - **Group** permissions
 - **Other** permissions (everyone on the system)
- Permissions are
 - **r** - read access
 - **w** - write access
 - **x** - execute

chmod [who][op][what] filename

- Can change permissions of a file using the **chmod** command
- who can be:
 - **u** - user permissions
 - **g** - group permissions
 - **o** - other permissions
 - **a** - all permissions
- op can be:
 - **+** - grant permissions
 - **-** - remove permissions
- what is one of the three permission types (**r,w,x**)

- The Filesystem
- Navigating the filesystem
- Users & Permissions
- Working with files
- Archives

Common Tasks

- create a file or directory
 - move a file or directory
 - copy a file or directory
 - deleting a file or directory
 - look at the contents of a file
 - look at the beginning of a file or end of a file
 - find a line of text in a file

File Creation

- A file can be created using:
 - **touch** <filename>
- Multiple files can be created by giving a list:
 - **touch** {a,b,c}
- Directories are created using:
 - **mkdir** <filename>
- We can create multiple directories in the same was as files:
 - **mkdir** {dir1,dir2,dir3}
- We can create a set of nested directories using the **-p** option. This creates all the directories not present:
 - **mkdir -p** this/is/a/new/directory

```
2. gareth@brutha: ~/Examples2 (ssh)
gareth@brutha:~/Examples2$ ls
gareth@brutha:~/Examples2$ touch a
gareth@brutha:~/Examples2$ ls
a
gareth@brutha:~/Examples2$ touch {b,c,d}
gareth@brutha:~/Examples2$ ls
a b c d
gareth@brutha:~/Examples2$ ls -l
total 0
-rw-rw-r-- 1 gareth gareth 0 Jan 15 10:18 a
-rw-rw-r-- 1 gareth gareth 0 Jan 15 10:18 b
-rw-rw-r-- 1 gareth gareth 0 Jan 15 10:18 c
-rw-rw-r-- 1 gareth gareth 0 Jan 15 10:18 d
gareth@brutha:~/Examples2$ mkdir dir2
gareth@brutha:~/Examples2$ ls
a b c d dir2
gareth@brutha:~/Examples2$ mkdir {dir1,dir3}
gareth@brutha:~/Examples2$ ls
a b c d dir1 dir2 dir3
gareth@brutha:~/Examples2$ mkdir -p this/is/a/new/directory
gareth@brutha:~/Examples2$ tree
.
├── a
├── b
├── c
├── d
└── dir1
    ├── dir2
    └── dir3
    └── this
        └── is
            └── a
                └── new
                    └── directory
8 directories, 4 files
gareth@brutha:~/Examples2$
```

Moving a File

- A file can be moved by:
 - **mv <src> <destination>**
- Where <src> and <destination> can be either absolute or relative paths.
- Multiple files can be moved by giving a list:
 - **mv {a,b,c,d} dir1**
- Or by using a file glob:
 - **mv dir1/* .**
- The move command can also be used to rename a file:
 - **mv a this_was_a**

```
2. gareth@brutha: ~/Examples2 (ssh)
gareth@brutha:~/Examples2$ tree
.
├── a
├── b
├── c
└── d
└── dir1

1 directory, 4 files
gareth@brutha:~/Examples2$ mv a dir1/
gareth@brutha:~/Examples2$ tree
.
├── b
├── c
└── d
└── dir1
    └── a

1 directory, 4 files
gareth@brutha:~/Examples2$ mv {b,c,d} dir1/
gareth@brutha:~/Examples2$ tree
.
└── dir1
    ├── a
    ├── b
    ├── c
    └── d

1 directory, 4 files
gareth@brutha:~/Examples2$ mv dir1/* .
gareth@brutha:~/Examples2$ tree
.
├── a
├── b
├── c
└── d
└── dir1

1 directory, 4 files
gareth@brutha:~/Examples2$ 
```

Copying a File

- A file can be copied by:
 - **cp** <src> <destination>
- Where <src> and <destination> can be either absolute or relative paths.
- Multiple files can be copied by giving a list or a glob:
 - **cp** {a,b,c,d} dir1
 - **cp** dir1/* .
- Copying directories is a little different, you need to tell copy that you want to recursively copy all the files, so:
 - **cp -r** dir1 dir2

```
2. gareth@brutha: ~/Examples2 (ssh)
.
├── a
└── b
    └── dir1

1 directory, 2 files
gareth@brutha:~/Examples2$ cp a dir1/
gareth@brutha:~/Examples2$ tree
.
├── a
└── b
    └── dir1
        └── a

1 directory, 3 files
gareth@brutha:~/Examples2$ cp {a,b} dir1/
gareth@brutha:~/Examples2$ tree
.
├── a
└── b
    └── dir1
        ├── a
        └── b

1 directory, 4 files
gareth@brutha:~/Examples2$ cp dir1 dir2
cp: omitting directory `dir1'
gareth@brutha:~/Examples2$ cp -r dir1 dir2
gareth@brutha:~/Examples2$ tree
.
├── a
└── b
    └── dir1
        ├── a
        └── b
    └── dir2
        ├── a
        └── b

2 directories, 6 files
gareth@brutha:~/Examples2$ 
```

Deleting a File

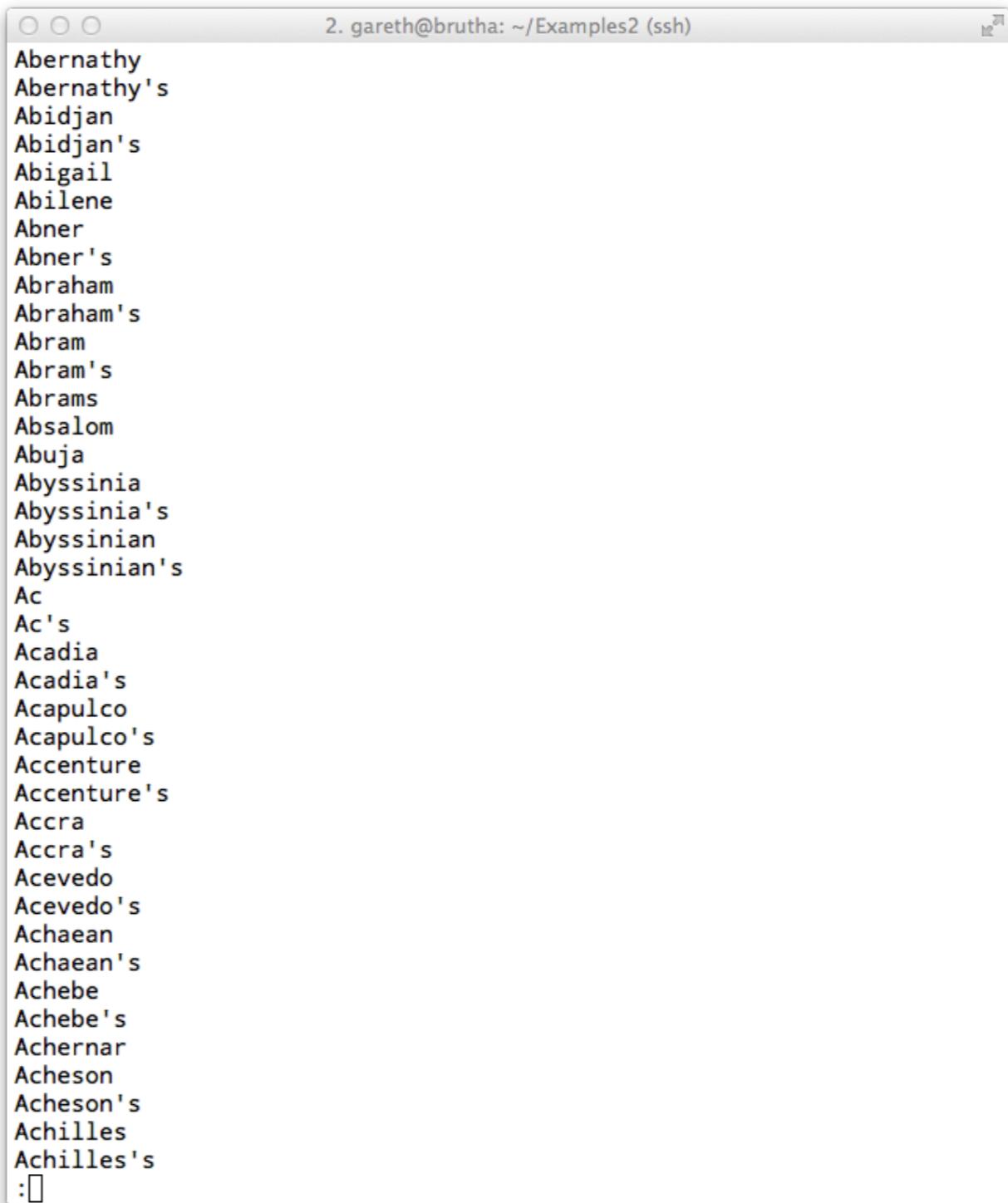
- A file can be deleted or removed by:
 - **rm <filename>**
- Where **<filename>** can be either an absolute or relative path.
- Multiple files can be removed by giving a list or a glob:
 - **rm {a,b,c,d}**
 - **rm dir1/***
- Like copying removing directories is a little different, you need to tell rm that you want to recursively delete all the files, so:
 - **rm -r dir1**
- Some files are also write protected (-r--r--r--) and you may have to remove with force:
 - **rm -f <filename>**
 - **rm -rf dir1**

```
2. gareth@brutha: ~/Examples2 (ssh)
gareth@brutha:~/Examples2$ ls
a b c d dir1
gareth@brutha:~/Examples2$ rm a
gareth@brutha:~/Examples2$ ls
b c d dir1
gareth@brutha:~/Examples2$ rm {b,c,d}
gareth@brutha:~/Examples2$ ls
dir1
gareth@brutha:~/Examples2$ rm dir1
rm: cannot remove `dir1': Is a directory
gareth@brutha:~/Examples2$ rm -r dir1/
gareth@brutha:~/Examples2$ ls
gareth@brutha:~/Examples2$ touch a
gareth@brutha:~/Examples2$ chmod -w a
gareth@brutha:~/Examples2$ ls -l
total 0
-r--r--r-- 1 gareth gareth 0 Jan 15 11:13 a
gareth@brutha:~/Examples2$ rm a
rm: remove write-protected regular empty file `a'? n
gareth@brutha:~/Examples2$ rm -f a
gareth@brutha:~/Examples2$ ls
gareth@brutha:~/Examples2$ 
```

CAUTION: When a file is removed it is gone, there is no wastebasket.

Looking at Files

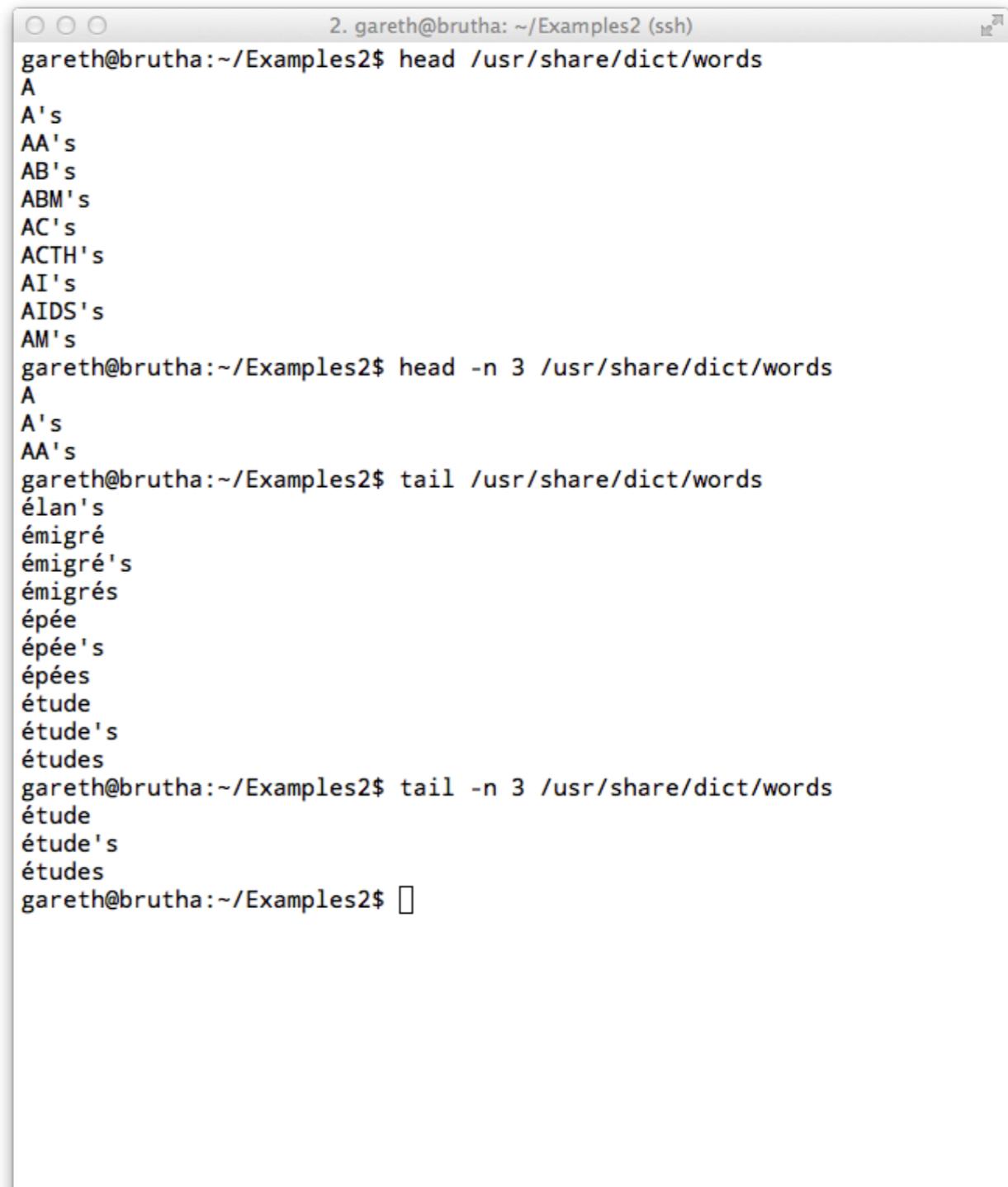
- They type of file can be found by using:
 - **file** <filename>
- The contents of file can be written to the terminal by using:
 - **cat** <filename>
- cat shows the whole contents of the file, we can see one page at a time using less or more:
 - **less** <filename>
 - **more** <filename>
- Of note, more will load the whole file into memory while less will only access the amount it needs to display.



A screenshot of a terminal window titled "2. gareth@brutha: ~/Examples2 (ssh)". The window displays a list of names and their possessives, likely from a file named "names.txt". The names listed are: Abernathy, Abernathy's, Abidjan, Abidjan's, Abigail, Abilene, Abner, Abner's, Abraham, Abraham's, Abram, Abram's, Abrams, Absalom, Abuja, Abyssinia, Abyssinia's, Abyssinian, Abyssinian's, Ac, Ac's, Acadia, Acadia's, Acapulco, Acapulco's, Accenture, Accenture's, Accra, Accra's, Acevedo, Acevedo's, Achaean, Achaean's, Achebe, Achebe's, Achernar, Acheson, Acheson's, Achilles, Achilles's, and a final colon followed by a blank line. The text is black on a white background, and the terminal has a standard OS X-style interface with a title bar and scroll bars.

Looking at bits of files

- For very large files (especially things like log file) you may only want to look at the start or end of the file.
- We can do this by using:
 - **head <filename>**
- This will print out the first ten lines of a file. If we want to print more or less than that we can specify, so:
 - **head -n 3 <filename>**
- As with head, we can also look at the end of a file with tail:
 - **tail <filename>**
 - **tail -n 3 <filename>**
- Tail can also be used to watch the end of a file and print out anything that is appended. This can be useful for watching things like logs:
 - **tail -f <filename>**

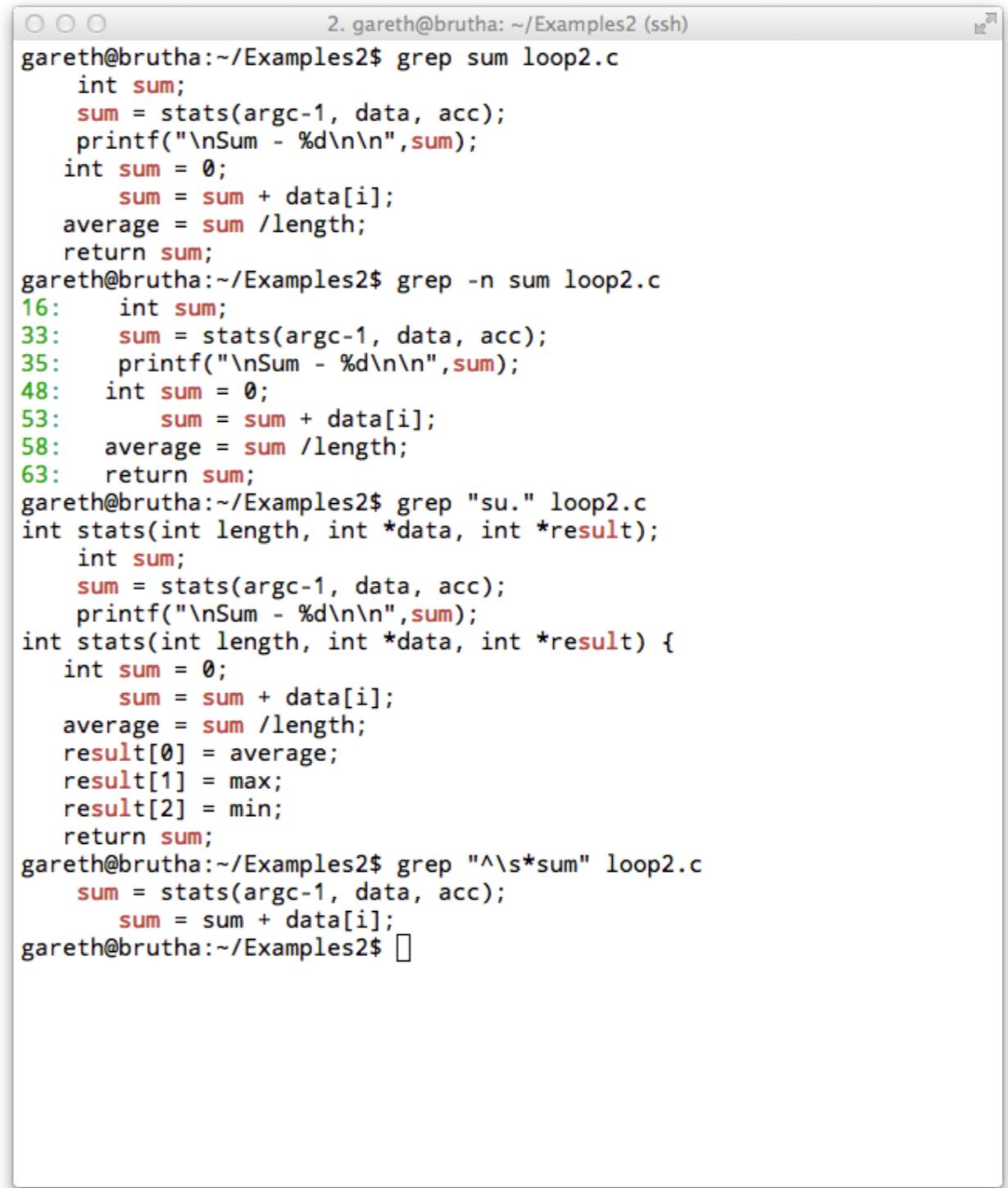


The screenshot shows a terminal window with three command examples:

- The first command, `head /usr/share/dict/words`, displays the first 10 lines of the /usr/share/dict/words file, which are mostly words ending in 's'.
- The second command, `head -n 3 /usr/share/dict/words`, displays only the first 3 lines of the same file.
- The third command, `tail /usr/share/dict/words`, displays the last 10 lines of the file, which are mostly words ending in 'e' or 'es'.
- The fourth command, `tail -n 3 /usr/share/dict/words`, displays only the last 3 lines of the file.

Searching in a File

- Often you'd like to search for a particular word or phrase in a text file (or a particular variable name in a piece of C code).
- We can do this by using the grep command:
 - **grep <expression> <filename>**
- grep takes something called a “regular expression” and searches for it through a file.
- To search for the word “sum” in a C program called loop.c, we would write
 - **grep “sum” loop.c**
- You can get the line numbers for each line that matches by using the -n option:
 - **grep -n “sum” loop.c**
- You can search for more complicated things by using patterns:
 - * - match 0 to as many of the preceding character
 - . - match exactly one character
 - [a-z] - match any lowercase character
 - [0-9] - match any digit
 - ^ - match the start of the line
 - \$ - match the end of the line

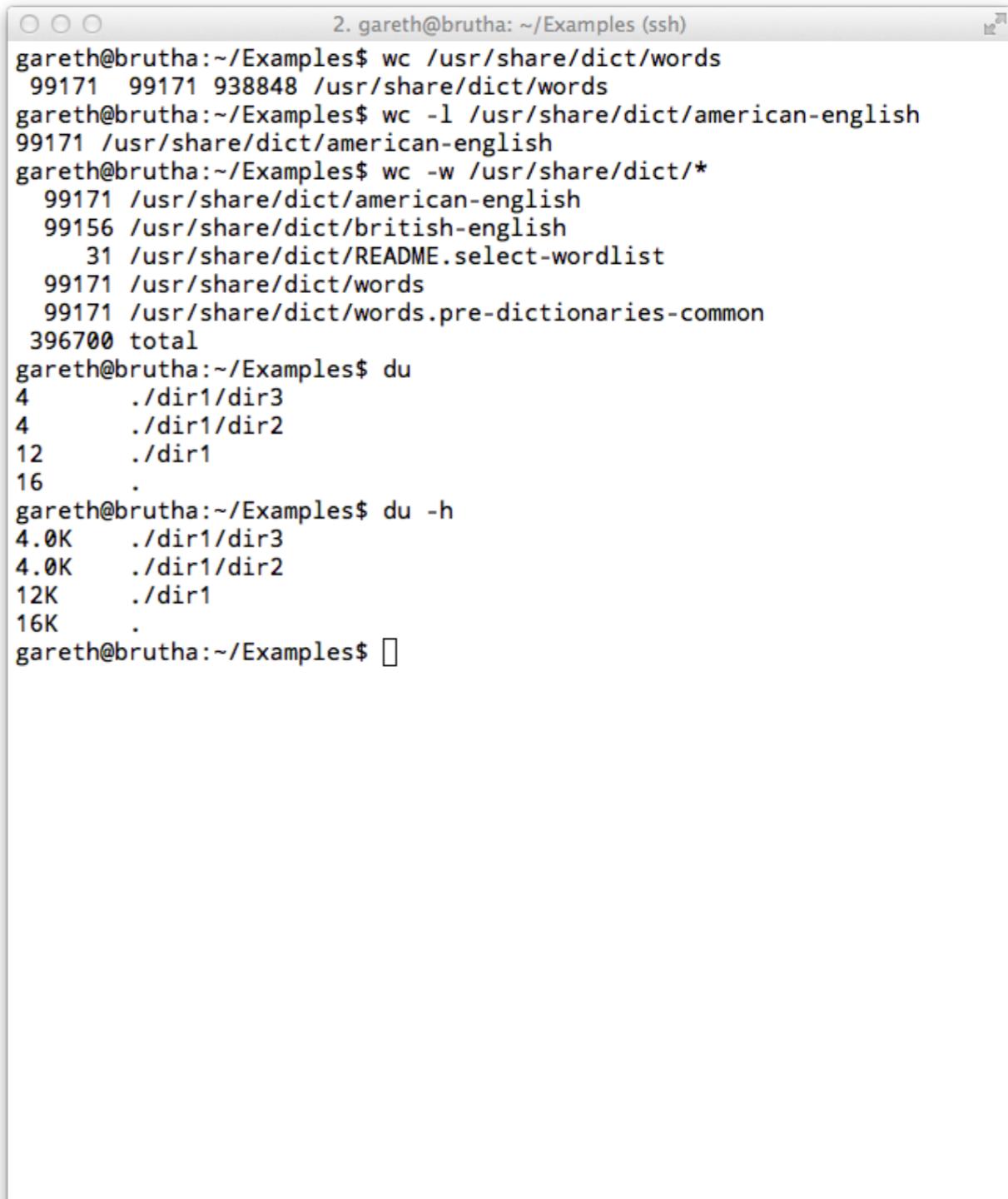


The screenshot shows a terminal window with the following session:

```
2. gareth@brutha: ~/Examples2 (ssh)
gareth@brutha:~/Examples2$ grep sum loop2.c
    int sum;
    sum = stats(argc-1, data, acc);
    printf("\nSum - %d\n\n",sum);
    int sum = 0;
    sum = sum + data[i];
    average = sum /length;
    return sum;
gareth@brutha:~/Examples2$ grep -n sum loop2.c
16:     int sum;
33:     sum = stats(argc-1, data, acc);
35:     printf("\nSum - %d\n\n",sum);
48:     int sum = 0;
53:     sum = sum + data[i];
58:     average = sum /length;
63:     return sum;
gareth@brutha:~/Examples2$ grep "su." loop2.c
int stats(int length, int *data, int *result);
    int sum;
    sum = stats(argc-1, data, acc);
    printf("\nSum - %d\n\n",sum);
int stats(int length, int *data, int *result) {
    int sum = 0;
    sum = sum + data[i];
    average = sum /length;
    result[0] = average;
    result[1] = max;
    result[2] = min;
    return sum;
gareth@brutha:~/Examples2$ grep "\^s*sum" loop2.c
    sum = stats(argc-1, data, acc);
    sum = sum + data[i];
gareth@brutha:~/Examples2$ 
```

Other Useful Commands

- Sometimes you want to find out how many words to lines are in a file. To do this use **wc**:
 - **wc <filename>**
 - **wc -l <filename>** (for lines)
 - **wc -w <filename>** (for words)
- You can sort the contents of a file using the **sort** command:
 - **sort <filename>**
- If you want to find out how much storage your files are taking up you can do:
 - **du <directory>**
 - **du -h <directory>**



```
2. gareth@brutha: ~/Examples (ssh)
gareth@brutha:~/Examples$ wc /usr/share/dict/words
 99171  99171 938848 /usr/share/dict/words
gareth@brutha:~/Examples$ wc -l /usr/share/dict/american-english
99171 /usr/share/dict/american-english
gareth@brutha:~/Examples$ wc -w /usr/share/dict/*
 99171 /usr/share/dict/american-english
 99156 /usr/share/dict/british-english
      31 /usr/share/dict/README.select-wordlist
 99171 /usr/share/dict/words
 99171 /usr/share/dict/words.pre-dictionaries-common
396700 total
gareth@brutha:~/Examples$ du
4      ./dir1/dir3
4      ./dir1/dir2
12     ./dir1
16     .
gareth@brutha:~/Examples$ du -h
4.0K   ./dir1/dir3
4.0K   ./dir1/dir2
12K    ./dir1
16K    .
gareth@brutha:~/Examples$ 
```

- The Filesystem
- Navigating the filesystem
- Users & Permissions
- Working with files
- Archives

Tarballs

- Sometimes you'd like to package up all your files in a single file.
- Similar to zipping or raring a file on Windows or OS X
- On Linux the default way for doing this is to create a “tarball”
- **tar** (tape archive) is a tool that was used for writing files to tape.
- It can be used to put all the files in a directory tree into a single file and preserve the paths.
 - **tar cvf <filename> <files>**
 - **tar tvf <filename>**
 - **tar xvf <filename>**
- You may also want to compress the archive, which can be done via tar or using gzip:
 - **tar zcvf <filename> <files>**
 - **gzip blob.tar**

```
2. gareth@brutha: ~/Examples (ssh)
gareth@brutha:~/Examples$ ls
a b c d dir1
gareth@brutha:~/Examples$ tar cvf blob.tar *
a
b
c
d
dir1/
dir1/dir3/
dir1/dir3/d
dir1/dir3/e
dir1/dir2/
gareth@brutha:~/Examples$ tar tvf blob.tar
-r--r--r-- gareth/gareth      0 2015-01-14 09:05 a
-rw-rw-r-- gareth/gareth      0 2015-01-14 09:05 b
-rw-rw-r-- gareth/gareth      0 2015-01-14 13:59 c
lrwxrwxrwx gareth/gareth      0 2015-01-15 08:56 d -> dir1/dir3/d
drwxrwxr-x gareth/gareth      0 2015-01-14 09:28 dir1/
drwxrwxr-x gareth/gareth      0 2015-01-14 09:06 dir1/dir3/
-rw-rw-r-- gareth/gareth      0 2015-01-14 09:05 dir1/dir3/d
-rw-rw-r-- gareth/gareth      0 2015-01-14 09:06 dir1/dir3/e
drwxrwxr-x gareth/gareth      0 2015-01-14 09:05 dir1/dir2/
gareth@brutha:~/Examples$ gzip blob.tar
gareth@brutha:~/Examples$ ls -l
total 8
-r--r--r-- 1 gareth gareth    0 Jan 14 09:05 a
-rw-rw-r-- 1 gareth gareth    0 Jan 14 09:05 b
-rw-rw-r-- 1 gareth gareth  262 Jan 15 12:07 blob.tar.gz
-rw-rw-r-- 1 gareth gareth    0 Jan 14 13:59 c
lrwxrwxrwx 1 gareth gareth   11 Jan 15 08:56 d -> dir1/dir3/d
drwxrwxr-x 4 gareth gareth 4096 Jan 14 09:28 dir1
gareth@brutha:~/Examples$ 
```

- The Filesystem
- Navigating the filesystem
- Users & Permissions
- Working with files
- Archives