UNIX files searching, and other interrogation techniques

- Ways to examine the contents of files.
- How to find files when you don't know how their exact location.
- Ways of searching files for text patterns.
- How to sort files.

William Knottenbelt (2001) http://www.doc.ic.ac.uk/~wjk/UnixIntro

File inspectors

Besides `cat` there are several other useful utilities for investigating the contents of files:

- `file` *filename(s)*
- `file` analyzes a file's contents for you and reports a high-level description of what type of file it appears to be:

```
    $ file myprog.c letter.txt webpage.html
    myprog.c:      C program text
    letter.txt:    English text
    webpage.html:  HTML document text
```

  file can identify a wide range of files but sometimes gets understandably confused (e.g. when trying to automatically detect the difference between C++ and Java code).

- `head, tail` *filename*
- `head` and `tail` display the first and last few lines in a file respectively. You can specify the number of lines as an option, e.g.

```
    $ tail -20 messages.txt
    $ head -5 messages.txt
```

  `tail` includes a useful `-f` option that can be used to continuously monitor the last few lines of a (possibly changing) file. This can be used to monitor log files, for example:

```
    $ tail -f /var/log/messages
```

  continuously outputs the latest additions to the system log file.

File inspectors

- `od` *options filename* (octal dump)
- `od` can be used to displays the contents of a binary or text file in a variety of formats, e.g.

```
$ cat hello.txt
hello world
$ od -c hello.txt
0000000  h   e   l   l   o       w   o   r   l   d \n
0000014
$ od -x hello.txt
0000000 6865 6c6c 6f20 776f 726c 640a
0000014
```

Finding files

There are at least three ways to find files when you don't know their exact location:

- **`find`**
- If you have a rough idea of the directory tree the file might be in (or even if you don't and you're prepared to wait a while) you can use `find`:

  `$ find` *directory* `-name` *targetfile* `-print`

  `find` will look for a file called *targetfile* in any part of the directory tree rooted at *directory*. *targetfile* can include wildcard characters. For example:

  ```
  $ find /home -name "*.txt" -print 2>/dev/null
  ```

  will search all user directories for any file ending in `".txt"` and output any matching files (with a full absolute or relative path). Here the quotes (`"`) are necessary to avoid filename expansion, while the `2>/dev/null` suppresses error messages (arising from errors such as not being able to read the contents of directories for which the user does not have the right permissions).

  `find` can in fact do a lot more than just find files by name. It can find files by type (e.g. `-type f` for files, `-type d` for directories), by permissions (e.g. `-perm o=r` for all files and directories that can be read by others), by size (`-size`) etc. You can also execute commands on the files you find. For example,

  ```
  $ find . -name "*.txt" -exec wc -l '{}' ';'
  ```

  counts the number of lines in every text file in and below the current directory. The `'{}'` is replaced by the name of each file found and the `';'` ends the `-exec` clause.

  For more information about `find` and its abilities, use `man find` and/or `info find`.

Finding files

- `which` (sometimes also called `whence`) *command*
- If you can execute an application program or system utility by typing its name at the shell prompt, you can use `which` to find out where it is stored on disk. For example:

```
    $ which ls
    /bin/ls
```

- `locate` *string*
- `find` can take a long time to execute if you are searching a large filespace (e.g. searching from `/` downwards). The `locate` command provides a much faster way of locating all files whose names match a particular search string. For example:

```
    $ locate ".txt"
```

  will find all filenames in the filesystem that contain "`.txt`" anywhere in their full paths.
  One disadvantage of `locate` is it stores all filenames on the system in an index that is usually updated only once a day. This means `locate` will not find files that have been created very recently. It may also report filenames as being present even though the file has just been deleted. Unlike `find`, `locate` cannot track down files on the basis of their permissions, size and so on.

Finding matching content in files

- `grep` (General Regular Expression Print)
- `$ grep` *options pattern files*

`grep` searches the named files (or standard input if no files are named) for lines that match a given pattern. The default behaviour of `grep` is to print out the matching lines. For example:

```
    $ grep hello *.txt
```

searches all text files in the current directory for lines containing "hello". Some of the more useful options that `grep` provides are:

`-c` (print a count of the number of lines that match), `-i` (ignore case), `-v` (print out the lines that don't match the pattern) and `-n` (printout the line number before printing the matching line). So

```
    $ grep -vi hello *.txt
```

searches all text files in the current directory for lines that do not contain any form of the word hello (e.g. Hello, HELLO, or hELlO).

If you want to search all files in an entire directory tree for a particular pattern, you can combine `grep` with `find` using backward single quotes to pass the output from `find` into `grep`. So

```
    $ grep hello `find . -name "*.txt" -print`
```

will search all text files in the directory tree rooted at the current directory for lines containing the word "hello".

Finding matching content in files

- The patterns that `grep` uses are actually a special type of pattern known as **regular expressions**. Just like arithemetic expressions, regular expressions are made up of basic subexpressions combined by operators.
  The most fundamental expression is a regular expression that matches a single character. Most characters, including all letters and digits, are regular expressions that match themselves. Any other character with special meaning may be quoted by preceding it with a backslash (\). A list of characters enclosed by '`[`' and '`]`' matches any single character in that list; if the first character of the list is the caret \``^`', then it matches any character not in the list. A range of characters can be specified using a dash (`-`) between the first and last items in the list. So `[0-9]` matches any digit and `[^a-z]` matches any character that is not a digit.
  The caret \``^`' and the dollar sign \``$`' are special characters that
  match the beginning and end of a line respectively. The dot '`.`' matches any character. So

```
    $ grep ^..[l-z]$ hello.txt
```

  matches any line in `hello.txt` that contains a three character sequence that ends with a lowercase letter from l to z.

Finding matching content in files

- `egrep` (extended grep) is a variant of grep that supports more sophisticated regular expressions. Here two regular expressions may be joined by the operator `` `|' ``; the resulting regular expression matches any string matching either subexpression. Brackets `'('` and `')'` may be used for grouping regular expressions. In addition, a regular expression may be followed by one of several repetition operators:

  `` `?' `` means the preceding item is optional (matched at most once).

  `` `*' `` means the preceding item will be matched zero or more times.

  `` `+' `` means the preceding item will be matched one or more times.

  `` `{N}' `` means the preceding item is matched exactly N times.

  `` `{N,}' `` means the preceding item is matched N or more times.

  `` `{N,M}' `` means the preceding item is matched at least N times, but not more than M times.

  For example, if `egrep` was given the regular expression

  ```
      '(^[0-9]{1,5}[a-zA-Z ]+$)|none'
  ```

  it would match any line that either:

  - begins with a number up to five digits long, followed by a sequence of one or more letters or spaces, or
  - contains the word `none`
- You can read more about regular expressions on the `grep` and `egrep` manual pages. Note that UNIX systems also usually support another `grep` variant called `fgrep` (fixed grep) which simply looks for a fixed string inside a file (but this facility is largely redundant).

Sorting lines

There are two facilities that are useful for sorting files in UNIX:

- `sort` *filenames*
- `sort` sorts lines contained in a group of files alphabetically (or if the `-n` option is specified) numerically. The sorted output is displayed on the screen, and may be stored in another file by redirecting the output. So

  ```
  $ sort input1.txt input2.txt > output.txt
  ```

  outputs the sorted concentenation of files `input1.txt` and `input2.txt` to the file `output.txt`.

- `uniq` *filename*
- `uniq` removes duplicate adjacent lines from a file. This facility is most useful when combined with `sort`:

  ```
  $ sort input.txt | uniq > output.txt
  ```

Exercises

1. Team up with your neighbor. Create a file called "`hello.txt`" in your home directory using the command `cat -u > hello.txt`.
   Ask your partner to change into your home directory and run `tail -f hello.txt`.
   Now type several lines into `hello.txt`. What appears on your partner's screen?
2. Use `find` to display the names of all files in the `/users/beerli` subdirectory tree.What do the errors mean (if any)?
3. Use `find` to display the names of all files in the system that are bigger than 1MB. Hint: use man find (-size)
4. Use `find` and `file` to display all files in the `/panfs/panasas1/users/beerli/simulations` subdirectory tree, as well as a guess at what sort of a file they are. Do this in two different ways.
5. Create a list of users presently logged in in the system: `finger > users.txt`
   Use `grep` to isolate the line in `users.txt` that contains your login details.
6. Use `grep` and `sort` to display a sorted list of all files in the `/panfs/panasas1/users/beerli/simulations` subdirectory tree that contain the word `“infile.1”` somewhere inside them.
7. Use `locate` to find all filenames that contain the word `emacs`. Can you combine this with `grep` to avoid displaying all filenames containing the word `share`?
8. Create a file containing some lines that you think would match the regular expression:
   `(^[0-9]{1,5}[a-zA-z ]+$)|none`
   and some lines that you think would not match. Use `egrep` to see if your intuition is correct. (use cat -u)
9. On Linux systems, the file `/dev/urandom` is a constantly generated random stream of characters. Can you use this file with `od` to printout a random decimal number?

3. Change to the home directory of another user directly, using `cd ~username`.
4. Change back into your home directory.
5. Make subdirectories called `work` and `play`.
6. Delete the subdirectory called `work`.
7. Copy the file `/etc/passwd` into your home directory.
8. Move it into the subdirectory `play`.
9. What is the difference between listing the contents of directory play with `ls -l` and `ls -L`?
10. What is the output of the command: `echo {con,pre}{sent,fer}{s,ed}`? Now, from your home directory, copy `/etc/passwd` and `/etc/group` into your home directory in one command given that you can only type `/etc` once.
11. Still in your home directory, copy the entire directory `play` to a directory called `work`, preserving the symbolic link.
12. Delete the `work` directory and its contents with one command. Accept no complaints or queries.
13. Experiment with the options on the `ls` command. What do the `d`, `i`, `R` and `F` options do?