CITS4407 Open Source Tools and Scripting
Text and regular expressions

Unit coordinator: Arran Stewart

Overview

This week:

- File expansion and globbing
- Regular expressions

Patterns in text

Often when using a system will want to be able to specify that we want to perform an operation on multiple files.

Patterns in text

Suppose we want to perform some operation on all files ending with "`.sh`" in the current directory – say, copy them to a backup directory.

We could write in all the filenames by hand:

```
$ cp myscript.sh myotherscript.sh first_assignment.sh ~/backups
```

Patterns in text

Suppose we want to perform some operation on all files ending with ".sh" in the current directory – say, copy them to a backup directory.

We could write in all the filenames by hand:

```
$ cp myscript.sh myotherscript.sh first_assignment.sh ~/backups
```

Or we could use a *wildcard character*, as follows:

```
$ cp *.sh ~/backups
```

Wildcards

The asterisk ("∗") represents an arbitrary string of characters; typing a command like `ls *.sh` will list all files ending in "`.sh`".[1]

(Other systems besides Unix-like ones use wildcard characters; Windows and MacOS X do, too.
And often, search engines or library-catalogue databases will allow users to include wildcard characters of some sort when searching for items.)

[1] Actually, it will not match files whose name starts with a full stop ("`.`"); they are called "dotfiles" and are Unix-like systems' equivalent of *hidden files* in Windows.

Filename expansion

When Bash sees a word containing $*$, it performs what's called *filename expansion*.

This process is also called *globbing*[2], and $*$, together with ? are known as *glob patterns*.

? matches a *single* letter in a filename:

```
$ ls
file1  file10  file2  file3
$ ls file?
file1  file2  file3
```

[2] See https://en.wikipedia.org/wiki/Glob_(programming) for an explanation of why it is called this.

Filename expansion

Another filename expansion lets us specify that a character must match one out of a set of characters we specify:

```
$ ls
file1  file10  file2  file3
$ ls file[23]
file2  file3
```

Going beyond files

Sometimes we will want to specify patterns in text other than filenames.

For instance, we might want to display ...

- all lines in a log file which start with the text "`ALERT:`"
- all lines in a file containing sales information, which contain the text "`keyboards`"
- all processes on the system started by a use who isn't us, and isn't `root`

Regular expressions allow us to do this.

What are regular expressions?

Regular expressions are a way of specifying *patterns* in text, and the `grep` command[3] gives us a way of searching for patterns in text.

The simplest sort of expression is just a sequence of ordinary characters.

`grep ALERT` will print all lines from standard input which contain the string "ALERT".

[3] Short for "**g**lobal **r**egular **e**xpression **p**rint".

Patterns in files

If we want to list all files in our current directory containing the word expression, we can do this using `grep`:

```
$ grep expression *
grep: lect01-images: Is a directory
lect03.md:(( ... )): (( expression ))
lect03.md:    Evaluate arithmetic expression.
```

`grep` options

Some useful options for the `grep` command:

- `-i` or `--ignore-case`: do a *case-insensitive* search
- `-v` or `--invert-match`: print all lines that *don't* match the regular expression.
- `-l` or `--files-with-matches`: list all files with lines that match the regular expression, but not their contents.

Metacharacters

The following characters, called *metacharacters*, have a special meaning to `grep`:

```
^ $ . [ ] { } - ? * + ( ) | \
```

We will look at each in turn and what they do.