

# CITS4407 Open Source Tools and Scripting

## Shell functions and script design

Unit coordinator: Arran Stewart

# Overview

This week:

- Regular expressions

## Regular expressions, cont'd

# Regular expressions

We've mentioned one tool that makes use of regular expressions, `grep`: it looks for lines in a file that match some pattern.

```
$ ls /bin | grep ^b
bash
btrfs
bunzip2
busybox
bzipcat
bzipcmp
bzdiff
bzegrep
bzexe
# ... more lines omitted
```

# Special characters in regular expressions

We said that the following characters, called *metacharacters*, have a special meaning to grep:

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

## Special characters in regular expressions

We've seen that the caret character, “^” means “at the start of the line” – `grep ^b` means, “print all lines beginning with the character b”.

Other special characters:

- “\$” means “at the end of the line”
  - `grep s$` means ‘print lines ending with the character “s”’.

```
$ ls /bin | grep 's$'
btrfs
bzless
less
ls
ps
ss
zless
# ... more lines omitted
```

# Special characters in regular expressions

```
$ ls /bin | grep 's$'  
btrfs  
bzless  
less  
ls  
ps  
ss  
zless  
# ... more lines omitted
```

(Note the use of single quotes – why might we use them?)

## Special characters in regular expressions – \$

What regular expression would match *only* the string “ls”?



## Special characters in regular expressions – .

A full stop matches any one character.

- `grep c.t` means 'print lines containing "c", some other character, then "t" '.

```
$ ls /bin | grep 'c.t'  
bzcat  
cat  
netcat  
ntfscat  
ntfsfallocate  
ntfstruncate  
zcat
```

## Special characters in regular expressions – \*

- An asterisk (“Kleene star”) means “zero or more of the previous item”

```
$ ls /bin | grep '^b.*'
bash
btrfs
bunzip2
busybox
bzipcat
bzipcmp
bzipdiff
bzegrep
bzxexe
# ... more lines omitted
```

So `grep '^b.*'` should actually give us identical results to `grep '^b'` – why?

## Special characters in regular expressions

Square brackets match a set or range of characters.

- `ls /bin | grep '^[bcd]'` finds commands starting with b, c or d.
- `ls /bin | grep '^[b-d]'` does the same.

## Special characters in regular expressions

Originally, grep only used the metacharacters we've mentioned, but later, others were added:

- | – match one thing OR another
- ? – match zero or one times
- + – match one or more times
- {n} – (where n is a number) match exactly n times
- {n,} – match n or more times
- {,n} – match at most n times
- {n,m} – match from n to m times

## Special characters in regular expressions

Because they were added later, these metacharacters are treated a bit differently. To use them we need to either

- use `grep` with the `-E` option, meaning “use *extended* regular expressions”, or
- put a backslash in front of the special characters, so `grep` knows they have a special meaning.

# Examples

- How can we find commands in `/bin` whose names are exactly three letters long, and start with “c”?

# Examples

- How can we find commands in `/bin` whose names are exactly three letters long, and start with “c”?
- How can we find commands in `/bin` whose names are exactly four letters long, and end with a letter from the range “d” through “g”?

# Examples

- How can we find commands in `/bin` whose names are exactly three letters long, and start with “c”?
- How can we find commands in `/bin` whose names are exactly four letters long, and end with a letter from the range “d” through “g”?
- How can we find commands in `/bin` whose names match *either* of the criteria above?



## sed – the “stream editor”

We can think of `grep` as being a little like the “find” functionality in a word processor or browser – it finds lines matching a pattern.

Is there an equivalent of “find and replace”?

# sed

There is – sed, the “stream editor”.

grep takes a single pattern to search for.

But sed takes two: a pattern to search for, and a string to replace it with.

## sed

```
$ ls /bin | grep '^c' | sed 's/^c/d/'  
dat  
dhac1  
dhgrp  
dhmod  
dhownd  
dhvt  
dp  
dpio
```

## sed

```
$ ls /bin | grep '^c' | sed 's/^c/d/'  
dat  
dhac1  
dhgrp  
dhmod  
dhownd  
dhvt  
dp  
dpio
```

The “s” means to search for the regular expression `^c`, and replace it with the letter `d`.

## sed

Conventionally, the forward slash ("/") is used to separate patterns, but we can use any character – handy if the forward slash turns up within the text we're trying to match.

```
$ ls /bin | grep '^c' | sed 's|^c|d|'  
dat  
dhacł  
dhgrp  
dhmod  
dhowñ  
dhvt  
dp  
dpio
```

## sed

```
$ ls /bin | grep '^c' | sed 'sZ^cZdZ'  
dat  
dhacĹ  
dhgrp  
dhmod  
dhowñ  
dhvt  
dp  
dpio
```

For clarity, it's usually best to stick to “/” or “|”.

You can also add g at the end to mean “search and replace, multiple times”.

- `sed s/aa/bb/g`

## Functions with regexes

Can we write a function which gives us a new command, `extension-rename`, which looks for files matching some particular extension, and renames them so they have another?

# Functions with regexes

A start: let's just print files matching some extension.

myfunctions.sh

```
extension_rename () {  
    orig_ext=$1  
    new_ext=$2  
    for file in *.${orig_ext}; do  
        echo $file;  
    done  
}
```

How does this work?



# Functions with regexes

Trying it out:

```
$ source myfunctions.sh  
$ extension_rename pdf  
lect01.pdf  
lect04.pdf  
lect05.pdf  
lect06.pdf
```

# Functions with regexes

Now let's print the new name, as well.

myfunctions.sh

```
extension_rename () {  
    orig_ext=$1  
    new_ext=$2  
    for file in *.${orig_ext}; do  
        new_file=`echo $file | sed "s/${orig_ext}/${new_ext}/"`;  
        echo $file $new_file;  
    done  
}
```

## Functions with regexes

Trying it out:

```
$ source myfunctions.sh
$ extension_rename pdf pdx
lect01.pdf lect01.pdx
lect04.pdf lect04.pdx
lect05.pdf lect05.pdx
lect06.pdf lect06.pdx
```

(There are actually ways of getting Bash to do what sed is doing here – look in the Bash Reference Manual, [§3.5.3 “Shell Parameter Expansion”](#). This would run faster, for very very large numbers of files; but we will stick to sed for the moment.)

## Functions with regexes

Finally, let's use the `mv` command to rename each file from the old name to the new name:

myfunctions.sh

```
extension_rename () {  
    orig_ext=$1  
    new_ext=$2  
    for file in *.${orig_ext}; do  
        new_file=`echo $file | sed "s/${orig_ext}/${new_ext}/"`;  
        mv $file $new_file;  
    done  
}
```

# Functions with regexes

Trying it out:

```
$ source myfunctions.sh
$ extension_rename pdf pdx
$ ls *pdf
ls: cannot access '*pdf': No such file or directory
$ ls *pdx
lect01.pdx  lect04.pdx  lect05.pdx  lect06.pdx
```

## Program size

In fact, we could put the body of our function in a script, `extension_rename.sh`, and use it that way:

```
extension_rename.sh
```

```
#!/bin/bash

orig_ext=$1
new_ext=$2
for file in *.${orig_ext}; do
    new_file=`echo $file | sed "s/${orig_ext}/${new_ext}/"`;
    mv $file $new_file;
done
```

The whole program takes 5 lines of code – small scripts are often shorter and more convenient to write than coding in Python, and *much* shorter than writing a program in a compiled language like Java or C.

## Other sed features

sed has many other features.

We can ask it to do a search and replace, but only on line 4, for instance:

```
sed '4s/dog/cat'
```

## Other sed features

For instance:

```
$ for ((i=0; i<5; i=i+1)); do echo $i dog; done | sed '4s/dog/cat/'  
0 dog  
1 dog  
2 dog  
3 cat  
4 dog
```

What is this doing?



## Search and replace in vi and vim

We will not cover it in detail, but vi and vim use much the same syntax for doing “search and replace”.

In either of these, the command

```
:%s/dog/cat/g
```

means “in the file we are currently editing – on all lines of the file, replace every occurrence of “dog” with “cat”.