University of Reading

UNIX INTERMEDIATE

A course building on the primer course for Unix operating system.

University of Reading

OBJECTIVE

- The objective of this course is to introduce some more advanced techniques in using the Unix operating system, covering more powerful commands and operators for filtering and processing data files.

- Prerequisites – you should be fluent in the use of basic Unix commands, and familiar with the filesystem.

Resource download:

https://github.com/paulmheaton/Unix-Intermediate

University of Reading

COURSE OVERVIEW

- Using grep to search file contents
- Using regular expressions
- Using the pipe(|) operator to chain outputs
- Using the find command
- Using awk

University of Reading

USING GREP TO SEARCH FILE CONTENTS

- The name *grep* means "general regular expression parser“
- Do not be put off by this mouthful – think of it as a search engine that can look inside files for bits of text.
- The simplest example of it’s use is to find a word or phrase in a bunch of files within a directory.
- Search within a single file:
- `grep ‘lamb’ my_poems/mary.txt`
- Search within a directory:
- `grep ‘lamb’ my_poems/*`

University of Reading

USING REGULAR EXPRESSIONS

- As mentioned earlier grep means "general regular expression parser“
- A regular expression is a collection of “codes” which tell grep how to search. These are very powerful and can be very complex, but we shall deal with a few simple examples here to give you a flavour of what they are about.
- Search for words that start (or contain) ‘m’ or ‘b’:
- `grep -l ‘[BM]ar’ my_poems/*`

- **Note the –l switch: this means only show the files which match. Omitting it will show all the files that grep has searched!**

University of Reading

MORE REGULAR EXPRESSIONS

- Search in files where the first word is Mary:
- `grep –l '^Mary' my_poems/*`

- A case insensitive search for Mary:
- `grep –il 'mary' my_poems/*`

- Search in files for sets of three digits:
- `grep –l '[0-9][0-9][0-9]' my_poems/*`

University of Reading

USING THE PIPE OPERATOR TO CHAIN OUTPUTS

- The ‘pipe’ operator is used to ‘chain’ the output of one command into another. Combined with grep (for example) it can be used to filter or generate new output.
- Try the following:
- `ls *.txt | grep ‘Mary’ my_poems/*`
- The thing to notice here is the pipe operator works from Right to Left, unlike the file operator(s) which work from Left to Right.
- The two operators can even be combined thus:
- `ls *.txt | grep ‘Mary’ my_poems/* > only_Mary_files.txt`
- You now have a file containing a list of files which only contain the word Mary!

University of Reading

USING THE FIND COMMAND

- It is often the case that you “loose” files in a sea of other files and directories and cannot find them!
- The find command will help you by showing you all the places where it finds a match for you file.
- The basic format is find [options] [tests] [actions] [filename expression]
- From your home directory (cd ~ if you are not in it) type:
- `find . –name barry.txt`
- Voila! find has searched through all the folders in your home directory and shows you where instances of barry.txt has been found.
- Notice the dot after the find command: this means search from the directory that you are in, otherwise you can put the path you want to search from here.

University of Reading

FIND ERRORS

- You will often see an error message printed by find something like:
- `find: ‘./.config/google-chrome’: Permission denied`
- This is because find tries to look through every directory at every file. If you do not have permission to read these files the error message is printed.
- To suppress these errors find allows you to redirect the output. Error reports have an intrinsic code of 2, so you can suppress outputs with this code by redirecting it to a null directory:
- `find . -name poem1.txt  2>/dev/null`
- This says “take any error messages (code 2) and redirect the output to a null directory so you only see the files found (a normal filename output is code 1).

University of Reading

FIND EXAMPLES

- `‘find . -name poem*.*` Find any file starting with “poem” in current and sub-directories.
- `find . -name *poem*.*` Find any file containing “poem” in the filename in current and sub-directories.
- `find /home -name *.jpg` Find all .jpg files in the /home and sub-directories.
- `find . -type f -empty` Find an empty file(s) within the current directory.
- `find /home -user your_username -mtime -7 -iname ".txt“` Find all txt files created by you within the last seven days, regardless of case.

University of Reading

USING AWK

- AWK is a command and a language for processing tabular data in a file or tabular data from the output of another command like ls –l.
- The basic format is: awk [options] ‘{action}’ file ..
- Download the file marks.txt and marks.csv into your home directory for the following examples.
- To see the entire table type:
- `awk '{print}' marks.txt`

AWK CONTINUED

- AWK automatically *parses* the file into a set of variables denoted by $n where n is the number of the column. By default AWK uses the spaces or tabs as a field separator, but you can change this.
- So if you just want to see the marks alone in the example type:
- `awk '{print $4}'`
- as the marks are in the 4th column. You should get:
- 80
- 90
- 87
- 85
- 89

MORE AWK

- To show more than one column simple add more variables to the print command, but ask awk to put some space between them.
- Type:
- `awk '{print $2 "  " $4}' marks.txt`
- Frank 80
- Rahul 90
- Betty 87
- Mary 85
- Peter 89

AWK CONTINUED

- AWK normally assumes that the field separator is a space(s) or a tab, but what if it is another symbol?
- AWK allows you to specify the Field Separator(FS) before processing the file thus:
- `awk '{FS=","; print $3 "  " $4}' marks.csv`
- Physics 80
- Maths 90
- Biology 87
- English 85
- History 89

AWK CONTINUED

- To print a running total of all the marks type: awk '{ total = total + $4 } {print tot}' marks.txt
- 80,170,257,342,431
- But what if you just want to know the total and not the running total?
- The END statement tells awk what to do when it has finished processing:
- Type: `awk '{ tot = tot + $4 } END {print “Total marks=“ tot}' marks.txt`
- Now you just get the last figure 431

TODO

THE END!

- Happy Bashing! ☺

- Please give us your feedback by following this link below:

  https://forms.office.com/Pages/ResponsePage.aspx?id=xDv6T_zswEiQgPXkP_kOX7ArvOm3cbpHnixhCNWKRS9UNjFCNjg2V1E1NkhSTldFUUFORFBRRzlXUy4u