

Warning: These notes are not complete, it is a Skelton that will be modified/add-to in the class. If you want to us them for studying, either attend the class or get the completed notes from someone who did

CSE2301

Grep—sed-awk

These slides are based on slides by Prof. Wolfgang Stuerzlinger at York University

1

Regular Expressions

- Regular expression (regex) describes a set of possible strings.
- Determines if there is a match or not
- Literal string matches itself
- Consider the regex **Fri**
- Regex can match string in more than one place

Today is Friday

Fri

Today is Thursday

2

Regular Expressions

- The . (dot) regular expression matches any character

Today is a nice day

c.

3

Regular Expressions

- “Character classes is used to match any one character of the set

`n[iea]ce`

Today is a nice day

- Negated character classes (no match)

`n[^iea]ce`

Today is a nice day

4

Regular Expressions

- `[Tt]`his matches This and this
- Can use ranges `[a-z]` or `[A-Z]` `[1-9]` `[a-e]`
- For portability across languages
 - `[a-zA-Z]` → `[:alpha:]`
 - `[a-zA-z0-9]` → `[:alnum:]`
- `^` beginning of line anchor
- `$` end of line anchor

`$n[iea]ce`

`^day`

`day$`

Today is a nice day

5

Repetition

- The `*` char matches the occurrence of the regular expression preceding it zero or more times
- `ab*c` matches **ac abc abbc abbbbc**
- `ab{n}` b repeated n times
- `ab{n,}` b repeated at least n times
- `ab{n,m}` b repeated at least n and at most m
- `ab{3,5}c` matches **abbbbc, abbbbc, abbbbbc**

6

Repetition

- character grouped using ()
- **xyz*** matches **xy, xyz, xyzz, xyzzz**
- **a(xyz)*** matches **a, axyz, axyzxyz, ...**
- **a(xy){2,3}** matches **axyxy and axyxyxy**

7

Grep

- Prints out all the lines in the input that **matches** an expression (quoted if whitespace).
- **grep** [options] **pattern** [file]
- Options let you do inverse search, ignore case,
- grep exits with 0 (found) 1 (not found) 2 (file not found)
- Regular expressions used in grep, sed, vi, awk to match a pattern

8

Variations of Grep

- fgrep – no regular expressions, only matches fixed strings
- egrep – or grep -E matches extended regular expressions
- for grep we use **\(, \), \[, and \}**
- for grep no escaping
- Do **man grep** for flags

9

Special characters in grep

- The shell can (and will) interpret special characters differently.
- You have to enclose the regex in single quotes to protect against this.
- Also you have to escape special characters in the regular expression if you want to match them literally
 - `'a*b'` matches **b, ab, aab, aaaab**
 - `'a\b'` matches **"a*b"**

10

Metacharacters

- Metacharacter is a character that represent something other than itself.

Metachar	Matches	Example	Meaning
^	Beginning of a line	'^here'	Here at the beginning of a line
\$	End of a line	'\$here'	Line ends with here
.	One character	'la..b'	A followed by 2 chars then b
*	Zero or more occurrence of last character	' *abc'	Zero or more spaces followed by abc
[]	One char in the set	'[L]ove'	Matches Love or love
[^]	One character not in the set	'[-A-K]Love'	Lines containing char not in A-K followed by ove

Metacharacters

Metachar	Matches	Example	Meaning
\<	Beginning of a word	'\<aa'	A word starts with aa
\>	End of word	'aa\>'	A word ends with aa
x\{m\}	X repeated m times		
x\{m,\}	X repeated at least m times		
x\{m,n\}	X repeated between m and n times		

12

Regular Expressions

- "foobar" matches (only) foobar
- '.' Matches any single character
 - f.obar matches faobar, fboar,
- [xyz] matches any character in the set
 - fo[abo]bar matches foabar, fobbar, foobar
- [^xyz] matches any character that is **not** in the set
 - fo[^ab]bar matches focbar, fodbar but not foabar

13

Regular Expressions

- '^' matches the beginning of a string
- '\$' matches the end of a string
- [a-z] matches any character in the range
- [0-9] matches any digit in the range
 - ^[ABC] matches A,B, or C at the beginning of a string
 - ^[^ABC] matches any character at the beginning of a string except A, B, and C
 - ^[^a-z]\$ matches any single character string except a lower case letter

14

Regular Expressions

- "\<" and "\>" matches the beginning and end of a word
- \"{n}\" matches n occurrences of the last char
- \"{n,}\" at least n occurrences
- \"{n,m}\" between n and m occurrences
 - ^A{4,8}B matches any line starting with 4,5,6,7, or 8 A's followed by a B
- ^([+|-]?[0-9]+\.?[0-9]*)*\$ **what is that?**

15

```

-123.24 that is a floating point number
786 that is an integer
Regular sentence
Another field
234.23
one sentence with one letter repeated twice in a row

tigger 259 % egrep 'let?er' test
tigger 260 % egrep 'let*er' test
one sentence with one letter repeated twice in a row
tigger 261 % egrep 'let+er' test
one sentence with one letter repeated twice in a row
tigger 262 % egrep 'let?er' test
tigger 263 % egrep 'one s[a-f]' test
one sentence with one letter repeated twice in a row
tigger 264 % egrep '^(+)?[0-9]+\.[0-9]*$' test
234.23
tigger 265 % egrep '^(+)?[0-9]+\.[0-9]*' test
-123.24 that is a floating point number
786 that is an integer
234.23
tigger 266 %

```

16

Regular expressions

- If you are looking for "the", [the] will match "other".
- you can insert space before or after, but that will be a problem if it is at the start or end of the line.
- We can use < and > to match words
- \<[tT]he\> does the trick

17

egrep

- Egrep: Additional regular expression metacharacters are added
- + (one or more proceeding char)
- ? zero or one proceeding char
- a|b either a or b
- () groups characters

18

fgrep

- Does not recognize any metacharacters as special characters

19

Examples

- variable names in C
 - `[a-zA-z][a-zA-Z0-9]*`
- Dollar amount with optional cents
 - `\$[0-9]+(\.[0-9][0-9])?`
- Time of day
 - `(1[012]|[0-9]):[0-5][0-9] (am|pm)`

20

GNU grep

- Linux uses the gnu version of grep
- Uses POSIX character classes
- GNU grep uses the extended set is with -E, even without -E it is available but we have to escape it
- Extended set “? + { } | ()”

21

POSIX Classes

Classes

[[:alnum:]]

[[:alpha:]]

[[:cntrl:]]

[[:digit:]]

[[:graph:]]

[[:lower:]] [[:upper:]]

[[:xdigit:]]

[[:punct:]]

Use

To use it `[[:alpha:]]`

Grep

`'[[:space:]]\.[[:digit:]]{3}[[:space:]'`

file

Other UNIX Utilities

- Uniq sort tr cut find awk (more later) xargs.
- `tr x y` # replace every occurrence of x by y
- `tr ab cd` #replace every occurrence of a by c and b by d
- `tr "[a-z]" "[A-Z]" <filename`
- `tr -s a <filename`

23

Other UNIX Utilities

- cut used to split data from files
- `cut [-ffields] [-ccolumn] [-dchar] filename`
- `cut -f3 -d, filename`
- `cut -c30-40 filename`
- `find . -type d -print`
- `find -type f -name "*.c" -print // or ' '`
-

24

Other Unix Utilities

- `xargs` `commands` execute the given command for each word in its stdin
- `find -type f -name "*.c" -print |xargs wc`
- `which prog`
- `whereis prog`
- `bg` and `fg`
- `Command &`
- `Command; command;`

25

UNIX Commands

- Grouping using `()` `date; who >temp`
- `(date; who) >temp`
- `>> file << pattern` Run command, if successfule run another command
- `Command && another command`
- `Command || another command`

26

Quotes

- Escape `\` is used to indicate the next character is not a special character.
- If a file name contains something like `'*'` `data*12`, we can refer to it as `data*12`
- We can use `' '` every character between these two single quotes is treated as non-special except `' cat 'data*12'`

27

Quotes

- ``` (back-quote) ``` the contents of the quote is treated as a shell command
- `echo `cat file``
- Double-quote `"` `"` like single quote except the variable substitution `$` and back-quotes ``` are still treated as special characters

28

Finally

- (command) is executed in a subshell
- `B=4` Set B=5 %% for csh
- `B=5`
- `echo $B` vs.
- `B=4`
- `(B=5)`
- `echo $B`
- Forking

29

Temporary Files

- Sometimes the script needs to create a temporary files, it should not have the same name as an existing one.

```
#!/bin/bash
```

```
newfile=$(mktemp newfileXXXX)
```

```
echo "Hello World" >${newfile}.1
```

Created 2 files newfile2468 and newfile2468.1

30

Example

- `#!/usr/bin/env bash`
- `# cookbook filename: trackmatch`
- `#`
- `for CDTRACK in *`
- `do`
- `if [["$CDTRACK" =~ "([[:alpha:]][:blank:]]*)-([[:digit:]]*)-(.*)$"]]`
- `then`
- `echo Track ${BASH_REMATCH[2]} is ${BASH_REMATCH[3]}`
- `mv "$CDTRACK" "Track${BASH_REMATCH[2]}"`
- `fi`
- `done`

bash 3.2

31

- Ludwig Van Beethoven - 01 - Allegro.ogg
- Ludwig Van Beethoven - 02 - Adagio un poco mosso.ogg
- Ludwig Van Beethoven - 03 - Rondo - Allegro.ogg

32

sed

- Sed: Stream editor is an editor to modify files.
- If you want to write a program to modify files, sed is the solution
- Here is a brief introduction to sed, practice is the best help.

33

sed

- `sed s/day/night <old >new`
- Substitute the word day in the file old by the word new and store the results in a file called new
- preferably `sed 's/day/night/'`
- If the string contains "/" then you have to escape it or use another delim.
 - `sed 's/\usr/local/bin/common/bin/' <old >new`
 - `sed 's_/usr/local/bin_/common/bin_' <old >new`

34

sed – Using &

- The special character & corresponds to the search pattern.
- For example to sed 's/[0-9]*/& &/' doubles a number at the beginning of a line
- "123 cat" → "123 123 cat"

35

- If you have many commands and they won't fit neatly on one line, you can break up the line using a backslash:

```
– sed -e 's/a/A/g' \  
    -e 's/e/E/g' \  
    -e 's/i/I/g' \  
    -e 's/o/O/g' \  
    -e 's/u/U/g' <old >new
```

36

- If you have a large number of *sed* commands, you can put them into a file and use
 - `sed -f sedscript <old >new`
- where *sedscript* could look like this:
 - # sed comment - This script changes lower case vowels to upper case
 - `s/a/A/g`
 - `s/e/E/g`
 - `s/i/I/g`
 - `s/o/O/g`
 - `s/u/U/g`
