Regular Expressions

- An algebra-like cryptic code for specifying text patterns
- Ubiquitous:
  - For all programming (&natural) languages and domains
  - Even search tools, both web and file, and many apps also…
  - Convenient for editing text… over global filesystem or corporation
    - from programming projects
    - to PR (Public Relations) as in case of new names, takeovers, rebranding etc
  - Programmable rather than error & repetitive strain injury prone GUI
- Powerful and expressive
- Portability issues
  - But not standardised…
    - Similarities but differences…
      - So easy to overlook and make mistakes
- Best done before deeper look at previous overview lect
  - On Find, grep, sed, awk leading on to bash programming.

Best way to learn is to use

But they are a bit of a drag… so brace yourself!

1) Get an intro and overview & study
   1) Lectures, labs & videos
2) get a Cheat sheet – true for anything
   1) Gives quick overview, and handy reference
3) Then mess with them
   1) Either with online tools
   2) Or using an editor, done in the labs … because
      1) Regex exercise
      2) Intro & exposure to programmable editors

Online tools

- Many others… this is just a selection
  - see resources on course site for more
- Probably the easiest for starters…
  - Mouseover context sensitive help
  - Easy to compare and make connections
    https://regexr.com/
- Probably the best for reference
  - https://www.freeformatter.com/regex-tester.html
- And it's always best to test before you go
  https://www.regextester.com/
- But remember the local implementation may differ!
  - Check it out on your local system – before committal!

Overview of character processing

- tr – basic character substitution – **tr**anslate
  - As an introduction to stream editing…
  - Stream!? … a stream of characters.. e.g. a file, pipe between processes etc.,
  - as distinct from interactive online editing…
  - Stream editing can be programmed and run in batch (non-interactive) mode … for large jobs on all files e.g. global updates to websites, after rebranding, takeovers
- Regexp – regular expressions – way to specify patters
  - Ubiquitous throughout CS… used everywhere, in

    interactive editors – ed, vi, vim, emacs … and derivatives…(incl Microsoft) etc.

    stream editors – sed

    Command arguments …e.g. ls *.txt

    SQL queries

    Language specification …although (E)BNF is used to specify grammars
- Grep family … a great find!? – actually finds often use grep
  - "**g**lobal **r**egular **e**xpression **p**rint" – but find is more meaningful!

grep Examples on a file: GrepText

For sample text, just redirect the manual on 'grep' to GrepText as a textfile to tes grep commands below, by issuing the command: (repeat with 'mat', not 'text')

man grep >GrepText

grep	'text '	GrepText	find text – will include context
grep	–w 'text '	GrepText	find the word text
grep	'\<text \>'	GrepText	again find the word (boundaries) 'text'
grep	'fo*'	GrepText	any of : f, fo, foo,
egrep	'fo+'	GrepText	any of : fo, foo, etc., but not just 'f'
egrep	-n '[Tt]ext'	GrepText	show lines + numbers with results'
grep	-nw '[Tt]ext'	GrepText	as previous but results are words
fgrep	'text'	GrepText	fgrep : fixed strings NOT regexs
egrep	'NC+[0-9]*A?'	GrepText	N, at least one C, any no. of digits, followed by an optional 'A' {0 or 1}
fgrep -f expfile		GrepText	search for strings from lines in file
fgrep -x 'for text'		GrepText	find lines with only 'for text'

TRanslate – tr – i.e. character translate

- simplest, so check if it will solve problem first
- Copies standard input to standard output with substitution or deletion of selected characters
- Syntax: *tr [-cds] [string1] [string2]*
  - *-c* - complements (everything except) the characters in *string1* with respect to the entire ASCII character set ..
  - *-d* - delete all input characters contained in *string1*
  - *-s* - squeeze all strings of repeated output characters that are in *string2* to single characters
- *tr* provides only simple text processing.
  - It does not allow the full power of *regular expressions*.
  - If you need the power of *regular expressions*, use *sed*

Examples

- *tr* reads from standard input.
  - Any character that does not match a character in *string1* is passed to *standard output* unchanged
  - Any character that does match a character in *string1* is translated into the corresponding character in *string2* and then passed to *standard output*
- Examples
  - *tr s z* replaces all instances of *s* with z
  - *tr so zx* replaces all instances of *s* with *z* and *o* with *x*
  - *tr a-z A-Z* replaces all lower case characters with upper case characters

- *string1* and *string2* can use ranges of characters as follows:
  - *tr a-z A-Z* translates all lower case to upper case
  - *tr a-m A-M* translates only lower case a through m to upper case A though M
- Ranges must be in ascending ASCII order
- What would happen if:

  tr 1-9 9-1

- *string1* and *string2* can use ranges of characters as follows:
  - *tr a-z A-Z* translates all lower case to upper case
  - *tr a-m A-M* translates only lower case a through m to upper case A though M
- Ranges must be in ascending ASCII order
- What would happen if:
  tr 1-9 9-1
- *tr* would interpret 9-1 as three individual characters since they are not in ascending ASCII order
  so if the numbers 1 through 9 were entered,
  tr would output 9, -, 1, 1, 1…….
- Actually – Linux & System V objects, BSD as above

Odd chars, Space & delimiter replace

- *tr* does a character by character 1-to-1 case sensitive swap
- Normal input output redirection and piping apply.
- This implies that certain characters must be protected from the shell (often called escape) by quotes or \, such as:
  - spaces : ; & ( ) | ^ < > [ ] \ ! newline TAB
- Example
  - *tr o ‘␣’* replaces all ‘o’s with a blank (space) typed but invisible
    common convention indicates a typed blank space as ‘␣’
  - tr ‘ ” ’ “ ‘ “ replace all double quotes with single ones
    in each case using other quote type as delimiter
  - tr \” \’ use backslash to escape quotes used as delimiters,
    and force literal interpretation of quote

Octal codes... begin with ‘\0’ ...for some chars

- Non-printing characters can also be specified

	Octal	Decimal	hexadecimal
Bell	‘\07’	7	7
Backspace	‘\010’	8+0 = 8	8
CR	‘\015’	8+5 = 13	d
Escape	‘\033’	3*8+3 = 27	1b = 16 + 11
Formfeed	‘\014’	8+4 = 12	c
Newline	‘\012’	8+2 = 10	a
TAB	‘\011’	8+1 = 9	9

Codes can also be expressed other bases…
Newline Octal 12 = ‘10’ decimal = ‘a’ in hexadecimal
CR : Octal 15 = 8+5 = ‘13’ decimal = ‘d’ in hex

Alternative to Octal codes... for some chars

\NNN character with octal value NNN (1 to 3 octal digits)

\\	backslash
\a	audible BEL
\b	backspace
\f	form feed
\n	new line
\r	return
\s	whitespace
\t	horizontal tab
\v	vertical tab

Newer character classes...Extended regex

[:alpha:]	*all letters*
[:upper:]	*all upper case letters*
[:lower:]	*all lower case letters*
[:digit:]	*all digits*
[:alnum:]	*all letters and digits*
[:graph:]	*all printable characters, excluding space*
[:print:]	*all printable characters, including space*
[:punct:]	*all punctuation characters*
[:blank:]	*all horizontal whitespace*
[:space:]	*all horizontal or vertical whitespace*
[:cntrl:]	*all control characters*
[:xdigit:]	*all hexadecimal digits*

- The *tr -d* option lets you delete any character matched in *string1*. *string2* is not allowed with the *-d* option
- Examples
  - *tr -d a-z* deletes all lower case characters
  - *tr -d aeiou* deletes all vowels
  - *tr -dc aeiou* deletes all character **except** vowels

    (note: this includes spaces, TABS, and newlines as well)
- Normal usage would be with pipes or redirection

  tr -d '\015' <in_file >out_file
  - Converts a DOS text file to a Unix text file by removing CR

    (DOS uses CR + newline/linefeed, Unix just uses newline/linefeed)

  (Reverse is not a deletion or 1-1 char map, so done with sed instead)
- Utilities often automatically convert now, and cmd line tools too.

M$ <> Unix/Linux line endings

While editors in 'nix & M$ display line endings correctly, the fundamental file line endings may remain unchanged, so bash and other programs may fail on this hidden catch. Worth highlighting, how to fix, if newer tools unavailable.

- Newer char class

  $ tr -d \\r < dosfile.txt > unixfile.txt

  the first '\' escapes the second '\' ;
  so taken literally as '\r', CR – carriage return
  (see a few slides back)
- Reverse IX 2 DOS
  - sed s/$/"\r"/ unixfile.txt > dosfile.txt
  - since tr does only 1 to 1, it cannot do 1 to 2, so cannot append CR to NL or LF (newline/linefeed)
  - quotes requirement depends on installation & configuration
- May be useful if old, minimal, version on old chip or minimal system (cheap IoT)

M$ <> Unix/Linux file conversion

DOS 2 IX : inbuilt MS ↔ IX file conv.

- Older – not on our system
  - dos2unix & unix2dos
- Newer replaces older
  - flip -u filename(s)
    - -u : to IX
    - -m : to MS

    will ensure that all are converted to target format, irrespective of initial format.
  - Common aliases include to (Unix/MicroSoft)
    - toix & toms

What does a Unix/Linux file really look like?

File as a byte stream … of ASCII here
Take a text file with the following 2 lines, spaces are tabs in 2nd line

0123456789abcdef
0123 4567 89AB CDEF

In a hex editor such as xxd (installed on CS servers) it looks like

Address	**ASCII codes as hexadecimal**	**Normal Text**
00000000:	3031 3233 3435 3637 3839 6162 6364 6566	0123456789abcdef
00000010:	0a30 3132 3309 3435 3637 0938 3941 4209	.0123.4567.89AB.
00000020:	4344 4546 0a	CDEF.

(All is hex except rightmost column, which shows characters)
A byte requires 2 hex chars, 4-bits each – 16 states 0-9a-f so 16 bytes/line displayed
Addresses are start address of line in hex, 16 bytes / line so 10 increments in hex (16)
ASCII code is stored in successive addresses in the file… ultimately in binary.
Decimal digits 0-9 are encoded as hex 30-39 (decimal 48-57)
A-F & a-f are encoded as hex 41-46 & 61-66 respectively (decimal 65-70 & 97-102)
Tabs and newlines are encoded as 09 and 0a respectively (decimal 9 and 10)
The hex file has addresses not lines, and what appears as the 3rd line begins at address 20H (32D)

How do Unix/Linux DOS/MS files compare?

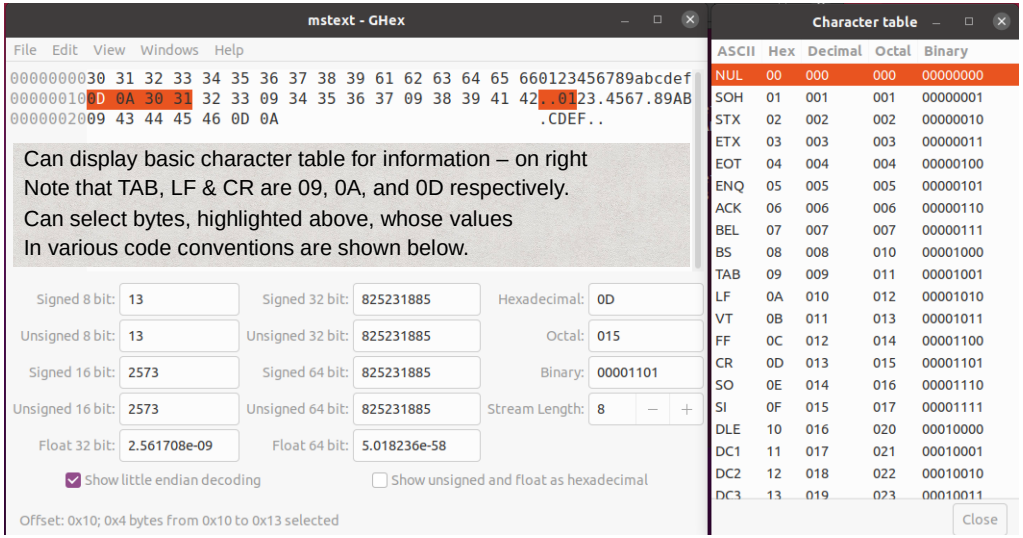
MS files include CR (carriage return) as well as LF (linefeed)

```
$ xxd ixtext
00000000: 3031 3233 3435 3637 3839 6162 6364 6566  0123456789abcdef
00000010: 0a30 3132 3309 3435 3637 0938 3941 4209  .0123.4567.89AB.
00000020: 4344 4546 0a                             CDEF.
```

```
$ xxd mstext
00000000: 3031 3233 3435 3637 3839 6162 6364 6566  0123456789abcdef
00000010: 0d0a 3031 3233 0934 3536 3709 3839 4142  ..0123.4567.89AB
00000020: 0943 4445 460d 0a                        .CDEF..
```

Both look identical under most tools & editors on both systems (Notepad & others on Win10 and on Ubuntu 20.04) which automatically convert, usually with some indication of underlying filetype such as DOS/Win or Unix text somewhere in the editor border,
but underneath clearly have different line endings may cause problems occasionally, especially on older minimalist systems.

Viewed in Ghex editor



Alternative uses of tr

- IoT / device monitoring – log files
  - Delete and squeeze repeated ‘ack’ chars
  - Indicating remote system still active
- Tidy up files from script session recording, using char classes as appropriate:
  - *[:blank:]* *all horizontal whitespace*
  - *[:space:]* *all horizontal or vertical whitespace*
  - *[:cntrl:]* *all control characters*

Example from tr man page

- How does this work?

tr -cs A-Za-z '𝒐012' <in_file > out_file

- It replaces all characters that
  - are not letters (-c for complement) with a newline (\012)
    sometimes \o is recognised as Octal represented 𝒐
  - then squeezes multiple newlines into a single newline (-s)
  - So basically outputs punctuation free words on separate lines

More tr Examples

- The following commands are equivalent
  - **tr 'abcdef' 'xyzabc'**
  - **tr '[a-c][d-f]' '[x-z][a-c]'**
- This command implements the "rotate 13" encryption
  - **tr '[A-M][N-Z][a-m][n-z]' '[N-Z][A-M][n-z][a-m]'**
- To make the text intelligible again, reverse the arguments
  - **tr '[N-Z][A-M][n-z][a-m]' '[A-M][N-Z][a-m][n-z]'**
- The -d option will cause tr to delete selected characters

echo If you can read this you can spot the missing vowels | tr -d 'aeiou'

gives f y cn rd ths y cn spt th mssng vwls

tr Gotchas

- Syntax varies between BSD and System V
  - BSD uses a-z
  - System V uses '[a-z]'
  - System V allows [*x***n*] to indicate *n* occurrences of *x*
    - If *n is omitted*, then *x* is duplicated as often as necessary
  - In BSD, if *string2* does not contain as many characters as *string1*, the last character of *string2* is duplicated as many times as necessary
  - System V fails to translate these additional characters – as does linux!
- Would *tr* solve the problem of capitalizing all first letters of each word in a sentence?

  Hint: tr only processes a single character – can't spot new word!?

  There are workarounds to show off, using recursion, arrays & shells; but neither relevant now nor pure application of 'tr'!

  And will be desperately slow whether recursive arrays, or starting shells

Shell recording of Caesar Cipher

When developing scripts at command line, it is much easier to test with sample text and modify as needed.

Here echo is used to feed a string to the tr command,

And both the text and command can be changed easily.

Online regex checkers work, but may not apply to your system!
So better look before you leap, as there may be no easy way back!?

$ echo 'Caesar cipher' | tr '[A-M][N-Z][a-m][n-z]' '[N-Z][A-M][n-z][a-m]'

Pnrfne pvcure

$ echo 'Pnrfne pvcure' | tr '[N-Z][A-M][n-z][a-m]' '[A-M][N-Z][a-m][n-z]'

Caesar cipher

Random / 1-liner text to test regex on your system

- Type & pipe it …
  - **echo 'test text here' |**
    - Clearly can easily modify test text to test regex
    - Easy on CLI with
      - line editing :
        - overtype to change
        - Quick moves: Ctrl A & E, Meta f & b, and left/right arrows
      - And history – if you need to go further back
- Or redirect output from
  - manual
    - **man cmd >testfile**
  - Or ls if working on filelists
    - e.g. **ls -lit >inodetimefilelist**
- And since these commands are filters taking stdio redirection can also be used for input
  - **tr <testfile**

Another easy source for various text

info bash -o- | shuf -n50 | sed 's/␣*/␣/g; s/^␣//' | fmt -w 90

- bash documentation from info
  - is output to stdout using the 'o' flag for info -o- ,
- piped it to shuf which randomly shuffles 50 lines,
- sed then
  - replaces multiple spaces (including none) with one space,
    - so words are s p a c e d o u t ! ! !
    - **s/␣*/␣/g**; => results in one space between non-whitespace chars:-
      - inserting one space where there was none, since * includes zero occurrences
      - and squeezing many spaces to one
    - **s/^␣//** strips the leading space at the start of a line
      - Whether remaining from multiple squeezed to 1
      - Or inserted where there was originally none
- finally **fmt -w 90** formats it to lines of max. width of 90.

Or go totally random

$ head -1 </dev/random >random_stuff

Which may appear like this on the screen, if >random_stuff is omitted:-

Or tee is used :- head -1 </dev/random | tee random_stuff

X45.I

Results in scattered lines & layout of non-printable character gobbledegook

(as it can backup both lines and screen!)

(- use Ctrl+L to clear the screen and find your prompt again)

So redirect to a file and use a hex editor

$ xxd random_stuff

```
00000000:  4c95 6fa4 6e69 f69e 604d 6c14 7e4d 7d74  L.o.ni..`Ml.~M}t
00000010:  6d6e 6b4c 2700 aaba 2105 853a f2d4 c7bb  mnkL'...!..:....
```

Which displays odd characters simply as '.' but can be further investigated using the char table

00 → NUL character

aa and ba are not defined in ASCII, but show up in Unicode as Latin-1 Supplemental female and male ordinal indicators – as in the following superscripts, 1st, 2nd, 3rd, etc for languages and nouns which have gender; as in 1ª prima donna or 1º primo uomo – as in operatic lead singers...

Or go totally random

tr -dc a-z0-9 </dev/random | tr 0-9 ' \n' | sed '/^\s*$/d'

- Similar - but different!
- Takes input from random number generator
  - /dev/random
    - with access to environmental noise,
    - (waits for enough entropy (measure of randomness) before operating)
- leaves only lowercase alphanumeric,
  - -dc a-z0-9 ==> dc => delete complement
- replaces decimal digits with newline : tr 0-9 '\n'
- deletes blank lines, sed '/^\s*$/d'
  - just using pattern match /^\s*$
    - where \s signifies whitespace.
  - Followed by d - delete

Printers random typesetting text – for ages…!

- Traditional favourite of typesetters for 100's years...tradition!
- lorem is a simple command-line wrapper around the "Text::Lorem" module, and generates the following number of
  - **p**aragraphs, **w**ords or **s**entences
- (package varies with distro, this is for Ubuntu)
  - $ lorem -p 3
  - $ lorem -w 5
  - $ lorem -s 1
- But is not currently installed on CS servers

Printers random typesetting text – for ages…!

Standard typesetting text for content-independent layout,

so that layout is not influenced by content !?

Apparently is garbled (typeset page-break?) extracts from work by Cicero (Roman orator & philosopher 1st century BC) entitled

"De Finibus Bonorum et Malorum"

"On the extremes (ends) of Good and Evil"

"Neque porro quisquam est, qui do**lorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed** quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem."

"Nor is there anyone who loves or pursues or desires to obtain pain of itself, because it is pain, but occasionally circumstances occur in which toil and pain can procure him some great pleasure."

Or simply:

No pain, no gain!?

Seems appropriate to inconsistent regex flavours!?

Line editor ***ed***: only for education

- ***ed*** : handy for showing regex and sed… & history
- OTHERWISE OBSOLETE : DO NOT USE …. UNLESS!
  THE FILES ARE MASSIVE & TOO BIG FOR NORMAL EDITORS
- ***ed*** - interactive old slow line editor …
  - a line at at time with regex
  - So slow you'd wouldn't know what is happening
- Unlikely - unless … societal collapse or apocalypse!!
  - Or low bandwidth audio hacks between machines!?
  - Designed for low RAM, bandwidth, CPU power and paper teletypes
  - Basic commands to print line ranges, and edit by substitution of characters or strings : s/this/that/
  - But it led to a visual editor :vi (but not as you know visual!)
- manual
  - https://www.gnu.org/software/ed/manual/ed_manual.html
- ***sed*** – the super batch (non-interactive) programmable stream editor is based on ed

Line editors for L-O-N-G files

- Line editors edit a line at a time,
  - Can do in a buffer in RAM, while rest of file on disk
  - So large files are easily edited
- Issues affecting in RAM file editing
  - configuration of RAM & swap file limits
  - Word size $2^{10} \sim 10^3$ = Kilo… going up in 1,000's

    Kilo, Mega, Giga, Tera, Peta, Exa, Zetta, Yotta
    - 32 bit = 2^{32} = 4 GB
    - 64 bit = 2^{64}= 16 Exabytes
  - Some upper limits for editors (often lower in practice)
    - Sublime Text 1 GB – practical limit 300MB
    - Notepad++ 2 GB
    - Emeditor ~ 250 GB (borrowing the em name from ed/ex/em)
      - First was ed, frills version em but hard on system,
      - Compromise was ex which led to vi