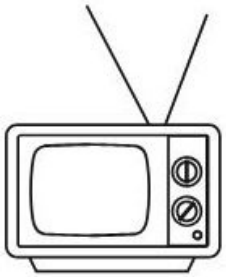
Announcements

Quizzes

- Limited # of them → drop the worst grade
- GQ-04 will be short and on M4

Case Study

- Order of presentation will be determined at random and announced during the sessions (in order to encourage attendance) If you are not there when called, you will lose points.
- Everyone should be ready to present during 1st session
- Presentations will last for 3-4 sessions during week #13 and #14



Previously On... CIS4930

- M1 – Basics
  - Aliases, filesystem, processes & signals
- M2 – Serious CLI
  - Shell quoting & escaping, Bash initialization files
  - Globbing, variables expansion, command expansion…
- M3 – Linux way
  - Redirections, piping, filters
- M4 – Regular Expressions
  - More filters: grep / egrep, Both Basic & Extended RegExs

Module

M04
Regular Expressions

Menu for this module

Regular Expression Families Considered:

- POSIX Basic RE (BRE)
- POSIX Extended RE (ERE)
- Perl Compatible RE (PCRE)

T1	Grep rudiments
T2	Grep Basic Regexs
T3	Character Classes
T4	Repeating patterns & capture groups
T5	Extended Regexs

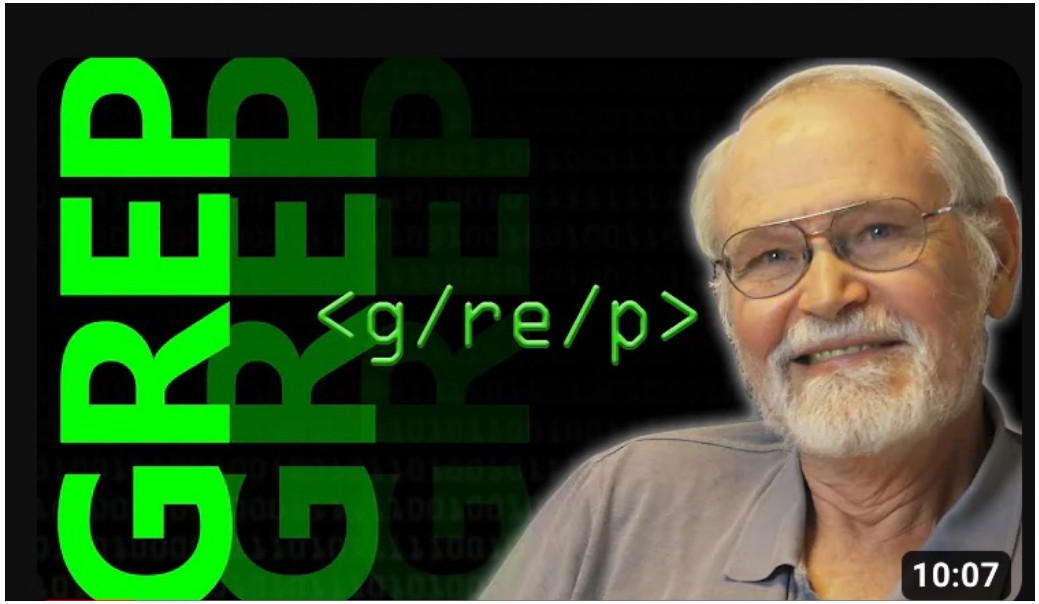
https://en.wikipedia.org/wiki/Regular_expression

Topic

M4T1
grep rudiments

https://youtu.be/-8v9k7lFlHI

First thing first… what does g/re/p mean?



https://youtu.be/NTfOnGZUZDk

Simple matches, e.g., single words

`grep information data.txt`

- The position of the word in the line is irrelevant
- We can match lines that have multiple occurrences of the word
- We get as output the whole line that contain the word

`grep -n information data.txt`

- Gives us also the line-number of each of the lines being matched

`grep -c information data.txt`

- Counts the number of lines matching w/o displaying them
- Equivalent to `grep information data.txt | wc -l`

By default, grep matches substrings

```
grep inform data.txt
```

- We match the same lines since “inform” appears inside “information”
- Same with "potato" and "potatoes" both being in data.txt

How can we match only “potato” or “inform”?

Matching strings that are words

```
grep -w inform data.txt
```

Word Separators are:

- Start of line
- End of line
- Space
- Tabulation

Matching strings that are alone on a line

We are no longer

- matching substrings within the line
- or even matching words within the line

→ The string has to be alone on the line!

```
grep -x "potato is here" data.txt
```

- We group the string in double quotes since it has spaces

Reverse Matches

```
grep -v "potato" data.txt
```

- As with previous options, this allow us to change significantly the behavior of grep and do what we would usually need regexs to do

Case insensitive matching

```
grep -i "potato" data.txt
grep -i "Potato" data.txt
grep -i "PoTaTo" data.txt
grep -i "pOtAtO" data.txt
```

Working recursively with folders

```
mkdir -p somedata/even-more-data/
gedit somedata/data.txt somedata/more.data.txt &
cp somedata/*.txt somedata/even-more-data/

grep -r "potato" somedata/
```

- Displays the name of the file where each match occurred
- Goes recursively through all subfolders

Displaying just the filenames w/ matches

```
grep -lr potato somedata/
```

- Lists only names of matching files (w/o the matching lines)
- The filenames do not repeat; each is displayed once only

```
grep -Lr potato somedata/
```

- We want the filenames in which there are no matches

Grep-ing in the wilderness

- Let's grep where we get lots of errors about file permissions

```
grep -irl warning /
```

- Let's now suppress these warnings

```
grep -sirl warning /
```

To sum it all up

Option	Meaning
`-w`	Matching words vs. substrings
`-x`	Matching words alone on a line
`-i`	Ignore cases
`-l`	List filenames with matches
`-L`	List filenames without matches
`-n`	Shows the line numbers
`-c`	Counts the matched lines instead of displaying them
`-v`	Display lines not matching
`-r`	Used on folders
`-s`	Suppress permission errors

Topic

M4T2
grep's Basic Regular Expressions

https://youtu.be/gMwpkse4OR8

Single Character Expressions

Syntax	Special Meaning
.	Any single character, except '\n'
[]	Any single character listed between the brackets
[^]	Any single character NOT listed between the brackets

```
something
homething
fomething
qomething
womething
```

```
grep 'something' datafile
grep 'omething' datafile
# matching only partial words!

grep '.omething' datafile
# now we match the whole word

grep '[sh]omething' datafile

grep '[shq]omething' datafile
grep '[qhs]omething' datafile
# order does not matter

grep '[^qhs]omething' datafile
```

Neutralizing meta-characters with \

Syntax	Special Meaning
\	Neutralizes the special meaning of the next character

```
Something else. And then more dots...
Homething again or \else...
Fomething else!!!
Qomething else???
Womething else569
[else]   three spaces
```

```
grep 'else' datafile

# we want else followed by .

grep 'else.' datafile
# → too many matches
grep 'else\.' datafile
# → OK

# we want else followed by …

grep 'else\...' datafile
# → KO: \ applies to 1 dot
grep 'else\.\.\.' datafile
# → OK
```

Another example

Syntax	Special Meaning
\	Neutralizes the special meaning of the next character

```
Something else. And then more dots...
Homething again or \else...
Fomething else!!!
Qomething else???
Womething else569
[else]   three spaces
```

```
#compare:
grep '[else]' datafile
grep '\[else\]' datafile
```

Neutralizing the neutralizer :)

Syntax	Special Meaning
\	Neutralizes the special meaning of the next character

```
Something else. And then more dots...
Homething again or \else...
Fomething else!!!
Qomething else???
Womething else569
[else]   three spaces
```

```
grep '\else' datafile

grep '\\else' datafile
```

Anchors

Syntax	Special Meaning
^	Beginning of line
$	End of line

```
Something else. And then more dots...
Homething again or \else...
Fomething else
else??? Qomething
Womething else
else]   three spaces
   else with 3 spaces before it
```

```
grep 'else' datafile
# → KO too many matches


grep '^else' datafile
grep 'else$' datafile
```

Anchors – more examples

Syntax	Special Meaning
^	Beginning of line
$	End of line

```
Fomething else
else??? Qomething
Womething else
else1
else2
else3
else2   three spaces
   else3 with 3 spaces before it
```

```
grep '^else' datafile
grep '^else$' datafile

grep '^else.$' datafile

grep '^else[23]$' datafile
# same as: grep -x
```

More Anchors: word delimiters

Syntax	Special Meaning
\<	Beginning of word
\>	End of word

```
Fomething else
else??? Qomething
Womething else
somethingelsegoesrighthere
else1
else2
else3
else2   three spaces
   else3 with 3 spaces before it
```

```
grep 'else' datafile
grep '\<else\>' datafile
# punctuation == separator too
```

Even more Anchors: more word delimiters

Syntax	Special Meaning
\<	Beginning of word
\>	End of word
\b	Word delimiter
\B	Anything but a word delimiter

```
grep '\(\belse\(3\|5\)\b\)\|something' datafile
grep '\b\(\(else\(3\|5\)\)\)\|something\)\b' datafile
```

Topic

M4T3
grep & character classes

https://youtu.be/BH2Jcdh2Gqw

Definitions

Syntax	Meaning	Example
`[:lower:]`	Matches lowercase letter	a-z
`[:upper:]`	Matches uppercase letter	A-Z
`[:alpha:]`	Matches upper- or lower-case letter	a-z A-Z
`[:alnum:]`	Matches upper, lower or digit character	a-z A-Z 0-9
`[:digit:]`	Matches a digit	0-9
`[:punct:]`	Matches a punctuation character	
`[:blank:]`	Matches whitespace (space, tab…)	

Examples

```
grep 'else[0123456789]' datafile

grep 'else[0-9]' datafile


grep 'else[[:digit:]]' datafile
# note the double [[ … ]]

grep 'else[[:digit:]][[:digit:]][[:digit:]]' datafile

grep '[[:alpha:]]omething' datafile
```

Note the
[[…]] pattern

Side Note: Shorthand Character Classes

ONLY with `grep -P` to enable PCRE (Perl Compatible RegExs)

Syntax	Meaning
\s	matches anything considered whitespace. This could be a space, tab, line break etc.
\S	matches the opposite of \s, that is anything which is not considered whitespace.
\d	matches anything considered a digit. ie 0 - 9 (It is effectively a shortcut for [0-9]).
\D	matches the opposite of \d, that is anything not considered a digit.
\w	matches anything considered a word character. That is [A-Za-z0-9_]. Note the inclusion of the underscore character '_'. This is because in programming and other areas we regulaly use the underscore as part of, say, a variable or function name.
\W	matches the opposite of \w, that is anything not considered a word character.

https://ryanstutorials.net/regular-expressions-tutorial/regular-expressions-intermediate.php#shorthand

Topic

M4T4
Repeating patterns & Capture Groups

https://youtu.be/8bsTCKWd8DY

Repeating patterns

Syntax	Meaning: Repeat previous RegEx…
`\{n\}`	…Exactly n times
`\{n,\}`	…N or more times
`\{0,m\}`	…M or fewer times
`\{,m\}`	…Same (not always supported)
`\{n,m\}`	…N to M times

Matches different digits being repeated
`else123`

```
grep 'else[[:digit:]][[:digit:]][[:digit:]]' datafile
grep 'else[[:digit:]]\{3\}' datafile
grep 'else[[:digit:]]\{1,\}' datafile
grep 'else[[:digit:]]\{1,3\}' datafile
```

Repeating patterns (including with *)

Syntax	Meaning: Repeat previous RegEx…
\+	…1 or more time
*	…0 or more times
\?	…0 or 1 time

Not the same meaning than * from globbing

```
grep 'else[[:digit:]]*' datafile
```

Capturing Groups

Syntax	Meaning
\(\)	Used to capture a group, i.e., a subpart of a regular expression

```
grep '\(else[[:digit:]]\)*' datafile
# NEW REQUIREMENT:
# do not just repeat the [[:digit:]]
# repeat whole pattern that was captured

grep 'else\([[:digit:]]\)\{3,\}' datafile

grep '\(else[[:digit:]]\{1,3\}\)\{1,\}' datafile
```

Same as before
else123

else12else123

Backreferring to captured groups

```
else5else5else5
Else3else3else3
elseelseelse
else1else2else3
```

Syntax	Meaning
`\(\)`	Used to capture a group, i.e., a subpart of a regular expression
`\n`	Backreference: Used to refer to a previously captured group

```
# group to repeat whole group:
grep 'elseelseelse' datafile
grep '\(else\)\{3\}' datafile

# repeat w/ different digits (as before):
grep 'else\([[:digit:]]\)\{3\}' datafile

# repeat w/ same digit:
grep 'else\([[:digit:]]\)\1\1' datafile
```

Alternatives

Syntax	Meaning
\\\|	Alternatives

```
else5else5else5
Else3else3else3
elseelseelse
else1else2else3
```

```
grep 'else\(3\|5\)' datafile
grep '\(else\(3\|5\)\)\|something' datafile
```

Topic

M4T5
Extended Regular Expressions

egrep or grep -E

https://youtu.be/3mLn7hrMGLk

Same old is the new "new"

```
else5else5else5
Else3else3else3
elseelseelse
else1else2else3
```

Syntax	Meaning
*	0 to n times
+	1 to n times
?	0 or 1 times
{n}	Exactly n times
{n,}	N times or more
{0,m}	At most m times
{,m}	At most m times
{n,m}	Between n and m times

```
egrep 'else[[:digit:]]?' datafile
egrep 'else[[:digit:]]*' datafile
egrep 'else[[:digit:]]+' datafile
egrep 'else[[:digit:]]{3}' datafile
```

{ } instead of \{ \}
+ ? Instead of \+ \?

Syntax	Meaning
*	0 to n times
+	1 to n times
?	0 or 1 times
{n}	Exactly n times
{n,}	N times or more
{0,m}	At most m times
{,m}	At most m times
{n,m}	Between n and m times

```
Something {else}
```

We do not need to backslash the curly braces, so () and {} are special characters, if you want them as just plain () or {} you'd need to backslash them

```
egrep '{else}' datafile
# KO, matches the } tho


egrep '\{else\}' datafile
```

( ) instead of \(\)
| instead of \|

Syntax	Meaning
`()`	Delimitates captured group
`\n`	Represent captured group #n
`|`	Alternatives
`\b`	Word boundaries (basic Regex)
`\B`	What \b does not match

```
else5else5else5
Else3else3else3
elseelseelse
else1else2else3
```

```
egrep '(else[[:digit:]])+' datafile

egrep '(else[[:digit:]])\1' datafile
egrep '(else[[:digit:]])\1\1' datafile

egrep 'else(3|5)' datafile
egrep '(else(3|5))|something' datafile

egrep '(\belse(3|5)\b)|something' datafile
egrep '\b((else(3|5))|something)\b' datafile
```