regular expressions workshop

LibrePlanet 2021
2021Mar20 @ 15:40 EDT
Virtual

der.hans
CDE
Object Rocket, a rackspace company
https://www.ObjectRocket.com/

Yes, we’re hiring :)

ObjectRocket
https://www.objectrocket.com/careers/

Rackspace Technologies
https://rackspace.jobs/

Upcoming Presentations

- List of Some Upcoming and Previous Talks and Publications
  - https://www.LuftHans.com/talks

Social Media and Fediverse

- FLOX_advocate on Mastodon
  - https://floss.social/@FLOX_advocate
- LuftHans on PLUME
  - https://fediverse.blog/~/LuftHans
- LuftHans on Freenode IRC
  - usually in #SeaGL, #LOPSA, #PLUGaz and #LibreLounge

Welcome to the Wonderful World of Regular Expressions

Presumptions

Participants have basic understanding of command line tools such as grep and echo.

Participants have basic understanding of shell features such as variables, pipes and output streams.

Lab requirements

Access to a bourne shell (/bin/sh or /bin/bash)

Introduction

Regular Expressions (RegEx) are sequences of characters that define a matching pattern using a specialized language.

RegEx define patterns that describe sets of strings. They are used by many common *NIX tools such as grep and sed. They can also be used in many programming languages such as Perl, PHP and Python. Database query languages also likely support RegEx.

Extended regular expressions are commonly available today and are easier to use than basic regular expressions. See the regex(7) man page, which is the regex page in section 7 of the manual, for more information. `man 7 regex` will show the man page.

Regular expressions were created in the 1950s.

Popularized by *NIX and the GNU tools :)

Things that are not RegEx

- Regular Expressions are not globs
- Both use similar characters for pattern matching, but in different ways
- Globs are evaluated by the shell **before** the command is run (generally)
- Regular Expressions are evaluated by the command (generally)
- Use quotes to protect RegEx from accidental globbification
- Globbing is **mostly** for filename matching
- RegEx is **mostly** for everything else
- RegEx are not explicitly limited to one line, but most tools do single line matching by default
- RegEx do not recurse, are not recursive
- RegEx are essentially state machines if you’re familiar with them

Lab setup

```
# Change to your $HOME directory.

cd

# Make a lab directory and change into it.

mkdir regexlab

cd regexlab

# Create some test files for the labs in the lab directory.

touch a aa ab one two three oooooo peel pole repeel file.htm file.html filehtml
```

Lab setup 2

```
# create a text file with some contents.
echo '  # comment with leading spaces

not a comment
  // double-slash style comment
Americans spell it color
British spell it colour
Meier is one way
Meyer is another way
Mexer is something else
# comment that starts at the beginning of the line' > file.txt
```

RegEx Examples

delete trailing white space

```
sed -re -i 's/ *$//' script.sh
```

search for dotted quad IP addresses somewhat sloppily

```
ip addr list | grep -E '(([12]{,1}[[:digit:]]{1,2})\.){3}([12]{,1}[[:digit:]]{1,2})'
    inet 127.0.0.1/8 scope host lo
    inet 192.168.0.42/24 brd 192.168.0.255 scope global dynamic eth0
```

search for MAC addresses

```
ip addr list | grep -E '([[:lower:][:digit:]]{2}:){5}[[:lower:][:digit:]]{2}'
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    link/ether 00:16:42:99:21:12 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Lab conventions

This lab presumes the use of **extended** regular expressions rather than basic regular expressions.

Command examples without example output will be just the command without indication of shell prompt.

```
ls -l /etc/passwd
```

Command examples with example output will begin with a dollar sign, **$**. Run the command given after the dollar sign.

After the command there will be example output followed by a final line with a dollar sign to indicate the end out ouput and a new prompt.

For instance:

```
$ ls -l /etc/passwd
-rw-r--r-- 1 root root 2030 2007-10-19 09:14 /etc/passwd
$
```

Lab conventions 2

This lab presumes you’re using some bourne shell derivative such as bash. If you’re using some sort of csh try running **`bash`** or **`sh`** to switch to a bourne shell.

This lab uses GNU tools. grep might have different behavior on *BSD (including Macs) or *NIX systems.

Use **`grep --version`** to determine which grep you’re using.

Tool: grep

grep (global regular expression print) searches for patterns in lines of text.

Search for a plain text string.

```
echo LibrePlanet 2021 | grep Libre
LibrePlanet 2021
$
```

Search for a plain text string without regard to case.

```
echo LibrePlanet 2021 | grep -i LIBRE
LibrePlanet 2021
$
```

Show just the matched part of the string.

```
$ echo LibrePlanet 2021 | grep -io LIBRE
Libre
$
```

Operator: dot

dot: `.`

A period, often called dot, represents any single character. It’s a mimic and similar to a wild card in poker. Dot matches exactly one character.

finds any files with at least 7 characters in the filename

```
$ ls | grep .......
file.htm
file.html
filehtml
file.txt
repeeled
$
```

match exactly one char; this matches files with `html` *in them, but not files that end in* `htm`

```
$ ls | grep htm.
file.html
filehtml
$
```

any character, it isn't a period; this matches `filehtml` *as well as* `file.html`, *also* `file-htmx` *if there was one*

```
$ ls | grep .htm.
file.html
filehtml
$
```

error-prone search for both `Meier` *and* `Meyer` *spellings of the name in a file*

```
$ grep Me.er file.txt
Meier is one way
Meyer is another way
Mexer is something else
$
```

Exercise: dot (5 minutes)

1. Use ls to list all files in the lab directory. Use grep to search output for filenames that are at least 5 characters long.
2. Use ls to list all files in the lab directory. Use grep to search output for filenames that have an e in them.
3. Use ls to list all files in the lab directory. Use grep to search output for filenames that have an e in them followed by at least two characters.

Operator: beginning of string anchor

beginning of string anchor: ^

aka: `hat`, `caret`

One of two `anchor operators` as it anchors the pattern to the beginning of the string being searched.

search for passwd lines with `bin` *in them, then use* `head` *to restrict output to the first 5 lines*

```
$ grep bin /etc/passwd | head -5
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
$
```

search for passwd lines that begin with `bin` *by anchoring to the beginning of the string*

```
$ grep ^bin /etc/passwd | head -5
bin:x:2:2:bin:/bin:/usr/sbin/nologin
$
```

search for two character account names, note the use of field delimiter to bound the match

```
$ grep ^..: /etc/passwd | head -5
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
```

Operator: end of string anchor

end of string anchor: $

aka: `dollar sign`, `EOS`

search passwd for accounts that have bash as the login shell

```
$ grep /bash$ /etc/passwd
root:x:0:0:root:/root:/bin/bash
lufthans:x:2112:2112:der.hans,,,:/home/lufthans:/bin/bash
$
```

search passwd for accounts that have a 4 character default command

```
$ grep /....$ /etc/passwd
root:x:0:0:root:/root:/bin/bash
sync:x:4:65534:sync:/bin:/bin/sync
lufthans:x:2112:2112:der.hans,,,:/home/lufthans:/bin/bash
$
```

Exercise: anchors (5 minutes)

1. Search `file.txt` for empty lines that have no content.
2. Search a `file.txt` for lines that start with a hash, `#`, to comment them out.
3. List all files in the lab directory. Search output for filenames that end in e followed by exactly one other character.

Operator: match zero or one

match zero or one: ?

aka: `question mark`

extended operator

This operator matches 0 or 1 of `whatever comes before it`.

It’s an extended RegEx operator, so use `-E` with grep.

Note: today `egrep` and `grep -E` should be the same. On my system, `egrep` is a wrapper that execs to run `grep -E`.

I recommend `grep -E` rather than `egrep`.

search filenames for those that end in `htm` *or* `html`

```
$ ls | grep -E html?$
file.htm
file.html
filehtml
$
```

search for both US and UK spellings

```
$ grep -E colou?r file.txt
Americans spell it color
British spell it colour
$
```

list files and search for those that start with `t` *and are 3 to 5 characters long*

```
$ ls | grep -E ^t...?.?$
three
two
$
```

Exercise: zero or one

1. list files and search for those that start with a and are 1 or 2 characters long
2. list files and search for those that have an o in either the first or second spot

RegEx varieties

Most command line tools understand extended RegEx, use extended RegEx when the tool supports them.

Many tools that use RegEx default to basic. Basic RegEx is still awesome, but requires more dilligence in escaping special characters.

Use PCRE (perl compatible regular expressions) or native matching for programming languages. Most languages that support RegEx can use PCRE.

Operator: plus

plus: +

This operator matches 1 or more of whatever comes before it.

Extended operator not in basic RegEx.

search filenames for those that have an o *followed by at least one other* o

```
$ ls | grep -E oo+
oooooo
$
```

search for lines that have been commented out, but start with at least one space

```
$ grep -E '^ +#' file.txt
  # comment with leading spaces
$
```

search filenames for those that have a `p` *followed by one or more* `e`*, then an* `l`

```
$ ls | grep -E 'pe+l'
peel
repeeled
$
```

Operator: star

star: `*`

aka: `splat`, `asterisk`

This operator matches zero or more of whatever comes before it. That includes the empty string in the case of zero of something. Regular expressions using star must be quoted on the command line to avoid shell interpretation.

Note: notice the quotes around the RegEx, we’ll get to those in a minute *that which comes before*

```
$ echo '*'
*
$ echo '*' | grep '*'
*
$ echo 'a' | grep '*'
$ echo '*' | grep '.*'
*
$ echo 'a' | grep '.*'
a
$ echo 'a' | grep 'a*'
a
$
```

Note: `.*` *also matches nothing as zero or more of anything includes nothing*

```
$ echo '' | grep '.*'

$
```

search for lines that have been commented out, but might start with spaces

```
$ grep -E '^ *#' file.txt
  # comment with leading spaces
# comment that starts at the beginning of the line
$
```

Quoting

All regular expressions used on the command line should be quoted to protect them from shell interpretation.

Some characters such as `*` and `?` are operators in both systems. They might be interpreted by the shell as globs rather than being handed off to the application.

Default to using single quotes, `'`. Use double quotes, `"`, if also needing some shell interpretation such as using a variable in the RegEx pattern.

Exercise: star (5 minutes)

1. list files, search for those have an e followed by
2. use star to emulate plus and search filenames for those that have a p followed by one or more e, then an `l`
3. search for lines commented out using a double-slash in `file.txt`

RegEx != Glob

Star is a workhorse for both regular expressions and globbing. For both it means `zero or more`, but in different ways.

In regular expressions, star matches zero or more of what came before it.

RegEx that searches for filenames starting with `o` *followed by zero or more* `o`

```
$ ls | grep -E '^oo*'
one
oooooo
$
```

In globbing, star matches zero or more in place.

glob that lists filenames that start with `oo` *followed by nothing or anything*

```
$ ls oo*
oooooo
$
```

remember to quote RegEx on the command line

```
$ ls | grep o*
$
```

using echo we can see the grep command that was run

```
$ ls | echo grep o*
grep one oooooo
$
```

Tool: sed

sed is the Stream EDitor. It can do many things.

Like grep, sed will search for a string. sed can also replace the found string with a new value.

This lab uses grep for search and sed for search and replace.

The s operator tells sed to search for the pattern in the first group and replace it with the value of the second group.

simple transform

```
$ echo tool | sed s/ol/ad/
toad
$
```

By default, sed matches the pattern once per line. Specifying g option at the end enables global so sed will match every time it finds the pattern.

lolling with g

```
$ echo tool | sed s/o/olo/g
toloolol
$
```

Like grep, sed expressions should be quoted. sed search and replace should also be used with some command line options.

The command line arguments I suggest for sed search and replace are `-r` (use extended regular expression) and `-e` (expression).

lolling with arguments and quotes

```
$ echo tool | sed -re 's/o/olo/g'
toloolol
$
```

Also, sed search and replace can use almost any character as the delimiter betwen the different parts of the expression, not just `/`.

Especially when are are slashed in the string or one of the parts of the expression, I usually use `@`.

using @

```
$ ls -d /etc/pa*s* | sed -re 's@/etc/@@'
papersize
passwd
passwd-
$
```

using ,

```
$ ls -d /etc/pa*s* | sed -re 's,/etc/,,'
papersize
passwd
passwd-
$
```

The delimiter doesn’t have to be punctuation, most any single character will do. Beware characters that might be interpreted by the shell or used in either the pattern or the replacement string.

using y

```
$ ls -d /etc/pa*s* | sed -re 'sy/etc/yy'
papersize
passwd
passwd-
$
```

grep returns matches, sed returns everything

```
$ ls | grep o | grep one
one
$ ls | grep o | sed -re 's/on/ZEBRA/'
ZEBRAe
oooooo
pole
two
$
```

Use `-n` to get quiet output from sed, but then add p to get matched lines

quiet, yet print

```
$ ls | grep o | sed -n -re 's/on/ZEBRA/p'
ZEBRAe
$
```

sed: many delimiters

```
$ ( for i in {a..b} {1..2} : . ? % ^ +; do echo "Trying $i - s${i}/vvv/${i}${i}"; echo
/vvv/yyz | sed -re "s${i}/vvv/${i}${i}"; done; )
Trying a - sa/vvv/aa
yyz
Trying b - sb/vvv/bb
yyz
Trying 1 - s1/vvv/11
yyz
Trying 2 - s2/vvv/22
yyz
Trying : - s:/vvv/::
yyz
Trying . - s./vvv/..
yyz
Trying ? - s?/vvv/??
yyz
Trying % - s%/vvv/%%
yyz
Trying ^ - s^/vvv/^^
yyz
Trying + - s+/vvv/++
yyz
$
```

sed: using RegEx

turn the first one or more o *into* ZZ

```
$ ls | sed -n -re 's/o+/ZZ/p'
ZZne
ZZ
pZZle
twZZ
$
```

turn the first two or more e *into* ZZ

```
$ ls | sed -n -re 's/ee+/ZZ/p'
pZZl
repZZled
thrZZ
$
```

the beginning of the string is the null string

```
$ ls | grep o | sed -re 's/e*/ZZ/'
ZZone
ZZoooooo
ZZpole
ZZtwo
$
```

as is the place between each of the characters in the string

```
$ ls | grep o | sed -re 's/e*/ZZ/g'
ZZoZZnZZ
ZZoZZoZZoZZoZZoZZoZZ
ZZpZZoZZlZZ
ZZtZZwZZoZZ
$
```

Behavior: greedy

The + and `*` operators are greedy in that they will match as many characters as they can.

Combined with `.`, + and `*` will greedily match as much as possible.

```
$ echo frederick | sed -re 's/e.+/EE/'
frEE
$
```

```
$ echo frederick | sed -re 's/e.*/EE/'
frEE
$
```

`.+` will not match the empty string, but `.?` and `.*` will.

```
$ echo '' | sed -re 's/.+/ZZ/g'

$ echo '' | sed -re 's/.?/ZZ/g'
ZZ
$ echo '' | sed -re 's/.*/ZZ/g'
ZZ
$
```

grep: inversion

grep can use the `-v` command line argument to invert the match and exclude a pattern from the output.

no e

```
$ ls | grep -Ev 'e'
a
aa
ab
oooooo
two
$
```

In this case, the `-v` tells grep to match anything that does not have an e in it.

Lab 1: RegEx thus far (20 minutes)

Lab setup, create a new lab dir and move to it, then create a few files.

```
mkdir regexlab-sd

cd regexlab-sd

touch one.txt one.log txt.log txt.txt
```

Exercises:

1. List the files, grep for those that have `txt` in the file names.
2. List the files, grep for those filenames that end in `txt`.
3. List the files, replace `txt` at the beginning of the filename with `tickets` showing only the files that were changed.
4. List the files, search for those that have a `t` in them and somewhere later an `o`.
5. List the files, convert the first `x` the filename to a `y`.
6. List the files, convert the first occurance of `ne` into the letter `y`.
7. List the files, convert a `t` along with the letter before the `t` and the letter after the `t` into a `y`.
8. List the files, convert a `t` along with the letter before the `t`, if there is a letter before the `t`, and the letter

   after the `t` into a `y`.

9. List the files, convert everything from the first `t` to the end of the filename to `ZZ`.
10. List the files, search for all that have an o followed by at least two characters, then convert the strings having `log` as a suffix to use `out` instead.

Before moving to the next section, change back into the main workshop directory.

```
cd ~/regexlab
```

Operator: single character quote

single character quote: `\`

aka: `backslash`, `upper-left to lower-right slash`

Having a period in filenames is a convenience for humans.

Still, we use them and in a regular expression a period in a string is seen as the dot operator leading to over-matching.

sometimes a `.` *is just a dot rather than the dot operator*

```
$ ls | grep -E '.html'
file.html
filehtml
$
```

Using the single character quote, the `.` can be escaped in the RegEx to be just a period.

Note: single character quote will be interpreted by the shell if the RegEx is not quoted.

now it’s just a dot

```
$ ls | grep -E '\.html'
file.html
$
```

Collections

The dot operator matches any character. Sometimes you want to match only a specific set of characters. Collections allow matching any character from a specific set.

Surrounding the group with inwardly facing square brackets inside a RegEx denotes a collection. Collections are also called `bracket expressions`.

match vowels

```
$ echo make | sed -re 's/[aeiou]/YZ/g'
mYZkYZ
$ echo tool | sed -re 's/[aeiou]/YZ/g'
tYZYZl
$
```

Combine with the `match previous character` operators to match multiple times in a row.

match multiple consecutive vowels

```
$ echo tool | sed -re 's/[aeiou]+/YZ/g'
tYZl
$ echo toad | sed -re 's/[aeiou]+/YZ/g'
tYZd
$ echo toadbookprize | sed -re 's/[aeiou]+/YZ/g'
tYZdbYZkprYZzYZ
$
```

match the first 3 letters of the alphabet followed by a vowel

```
$ echo 'aero car flies above ground' | sed -re 's/[abc][aeiou]/YZ/g'
YZro YZr flies aYZve ground
$
```

Character classes

When trying to match any letter rather than punctuation, this will require a lot of typing. And can lead to typos.

this pattern no `r`

```
$ echo 'toolZ 4ever' |  sed -r -e 's/[abcdefghijklmnopqqstuvwxy]/Y/g'
YYYYZ 4YYYr
$
```

We have character classes to help out. They can be used within a collection.

Character classes are enclosed in colons, which are enclosed in matching, inwardly facing, square brackets.

The classes have plain english names.

```
[:alpha:]
[:upper:]
[:lower:]
[:alnum:]
[:digit:]
[:punct:]
```

See regex(7), glob(7) and wctype(3) for descriptions of the character classes.

Note that the square brackets are part of the collection, so when using character classes there will be `at least` two sets of square brackets. There will be outer brackets for the collection and inside the collection there will be brackets on the character class.

```
$ echo 'toolZ 4ever' |  sed -re 's/[[:lower:]]/Y/g'
YYYYZ 4YYYY
$ echo 'toolZ 4ever' |  sed -re 's/[Z[:lower:]]/Y/g'
YYYYY 4YYYY
$ echo 'toolZ 4ever' |  sed -re 's/[[:alpha:]]/Y/g'
YYYYY 4YYYY
$ echo 'toolZ 4ever' |  sed -re 's/[5[:alpha:]]/Y/g'
YYYYY 4YYYY
$ echo 'toolZ 4ever' |  sed -re 's/[4[:alpha:]]/Y/g'
YYYYY YYYYY
$ echo 'toolZ 4ever' |  sed -re 's/[[:alnum:]]/Y/g'
YYYYY YYYYY
$
```

As a character class is just one entity in a collection, multiple character classes and other entities can be used at the same time.

eliminate numbers, spaces and vowels

```
$ echo 'l33t sp34k 4tw' | sed -re 's/[[:digit:][:space:]aeiou]//g'
ltspktw
$
```

Exercise: Collections and character classes (5 minutes)

1. Create a regular expression the will check to see if the input string ends in an odd number (hint: `echo $RANDOM` will give you a random number for the input)
2. Search `file.txt` for lines that have punctuation in them
3. Search `file.txt` for lines that have punctuation, a capital `A` or an `x` in them

Ranges

Collections can also be specified as ranges of character such as `a-z`, `0-9`, `A-F`.

Avoid using them as they are not portable across languages and character sets.

find files starting with the first 6 letters of the English alphabet in lower case

```
$ ls | grep -E '^[a-f]'
a
aa
ab
apples
file.htm
file.html
filehtml
file.txt
$
```

change letters to colons

```
$ echo 'UPPERlower' | sed -re 's/[a-zA-Z]/:/g'
::::::::::
$
```

Beware lexical issues due to localization. See equivalence classes in regex(7).

The ranges can be for punctuation, e.g. `,-:`, but that will be unuseful, use the `[:punct:]` character class or specify the specific punctuation you want to match.

In order to include a dash, `-`, in a collection, it must be listed first or last in order to avoid being seen as the range operator.

Back references

We can use parenthesis to save parts of the match for later use.

find and return just the first vowel

```
$ echo tool | sed -re 's/.*([aeiou]).*/\1/'
o
$ echo tool | sed -re 's/.*([aeiou]).*/\1 \1 \1/'
o o o
$
```

The digits 1 to 9 represent the first 9 back references.

Ampersand, &, represents the entire string matched.

the first vowel, the entire matched string, then the first vowel again

```
$ echo tool | sed -re 's/.*([aeiou]).*/\1 & \1/'
o tool o
$
```

introducing a spy

```
$ echo 'James Bond' | sed -re 's/.+[[:space:]]+([[:alpha:]]+).*/\1, &/'
Bond, James Bond
$
```

Branching

Sometimes you need this or that. Branching allows searching for alternatives.

Seperate alternatives with a pipe.

get details for root, bin and sync accounts

```
$ $ grep -E '^root:|^bin:|^sync:' /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
$
```

To embed the alternatives in a larger RegEx, surround them with parenthesis.

match the same thing as before, less typing

```
$ grep -E '^(root|bin|sync):' /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
$
```

Parenthesis matches are also back references.

grab username and UID for root and bin accounts

```
$ sed -n -re 's/^(root|bin):x:([[:digit:]]+):.*/\1 \2/p' /etc/passwd
root 0
bin 2
$
```

Specific Number of Matches

Use a number surrounded by inward facing curly braces to specify exact number of times to match.

drop three e

```
$ echo sleeeeep | sed -re 's/e{3}//'
sleep
```

Match specific minimum and maximum times. Minimum goes before the comma, maximum after, all within inwardly facing curly braces. Matches will be greedy.

```
$ echo sleeeeep | sed -re 's/e{3,5}/E/'
slEp
$ echo sleeeeep | sed -re 's/e{3,}/E/'
slEp
$ echo sleeeeep | sed -re 's/e{,3}/E/'
Esleeeeep
```

Not so greedy looking on that last one?

verify a file is read-only

```
$ ( for i in -r--r--r-- -r--r----- -r-----r-- -r--------; do echo "$i" | grep -E '^-r--
(r--|---){2}'; done )
-r--r--r--
-r--r-----
-r-----r--
-r--------
$
```

Note: please use `stat` instead.

Operator: inversion operator

inversion operator: `^`

aka: not

The inversion operator can be used in a RegEx collections to exlude characters rather than than including them. It’s similar to the `-v` inversion option for grep, but within RegEx collections.

leave only lower case letters

```
$ echo 'UPPER.:-lower' | sed -re 's/[^[:lower:]]//g'
lower
$
```

Note, that inversion, `^`, and range, `-`, are the only operators for within collections. Even backslash, `\`, and quotes are not special characters within a collection. Even so, the shell might interpret quotes before starting the program that uses the RegEx.

If wanting to invert the match on a collection, then the inversion operator must appear first in the collection.

If wanting to include a closing square bracket in a collection, then it must appear second behind the inversion operator or first before anything else if it not an inversion match.

A dash must be listed first or last in a collection to be the character rather than the operator.

Lab: end of workshop

1. Search `file.txt` for lines that have either `Meyer` or `Meier`
2. Search the filenames those that have two consecutive vowels in them
3. Search files.txt for lines that begin with either a space or a tab
4. Search files.txt for lines that do not begin with either a space or a tab
5. Search the filenames for those that don't have a period or an e in them
6. Search for filenames that don't have the first 5 letters of the English alphabet (remember that ASCII has 52 characters in the alphabet)

Lab: homework

Finish any labs not completed during the workshop.

Review the example RegEx at the beginning to understand why they match.

Let me know if you find bugs in the examples or elsewhere in the workshop.

Questions later?

Feel free to ask me follow up questions via Mastodon or IRC.

Resources

- RegEx creation history
- regex(7)
- wctype(3)
- short history of grep
- longer history of early days, including grep
- web site for learning, building and testing RegEx
- site for reviewing RegEx support and syntax in various tools and languages
- learn RegEx via crossword puzzles
- rgxg - command-line tool to generate regular expressions

Resources: egrep

`egrep` *is* `grep -E` *wrapper*

```
$ cat /bin/egrep
#!/bin/sh
exec grep -E "$@"
$
```