Regular Expressions & grep

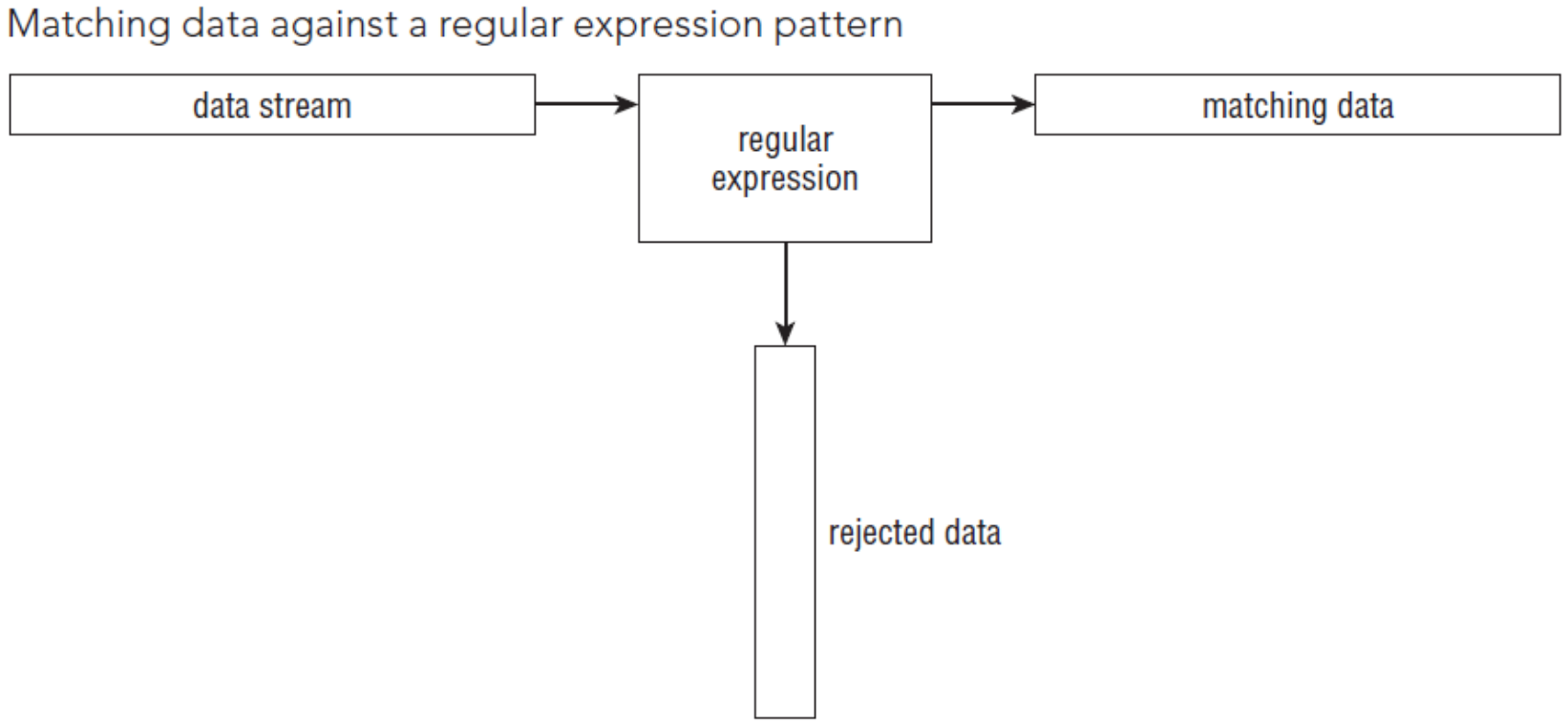
Dr. Vimal Baghel, Assistant Professor, SCSET, BU

Outline

- Regular Expressions
  - Definition, and Usage
  - Character classes
  - Anchors
  - repetitions
  - Subexpressions
- grep
  - Grep family

What is Regular Expression?

- A *regular expression* is a pattern template we define that a Linux utility (sed/awk) uses to filter text.
- A Linux utility matches the regular expression pattern against data as that data flows into the utility.
  - If the data matches the pattern, it's accepted for processing.
  - If the data doesn't match the pattern, it's rejected.



What is regex?

BENNETT UNIVERSITY
THE TIMES GROUP

- A *regular expression* (*regex*) is a rule that a computer can use to match characters or groups of characters within a larger body of text.
- regex is a set of possible input strings, descended from finite automata theory
  - For instance, using regular expressions, you could find all the instances of the word *cat* in a document, or all instances of a word that begins with *c* and ends with *t*.
- *regex is used in*
  - **vi**, **ed**, **sed**, and **emacs**
  - **awk**, **tcl**, **perl** and **Python**
  - **grep**, **egrep**, **fgrep**
  - **compilers**

Regular Expressions

BENNETT UNIVERSITY
THE TIMES GROUP

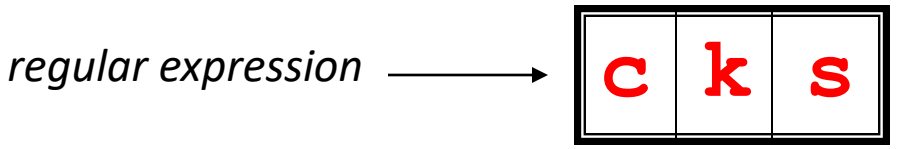
- The simplest regular expressions are a string of literal characters to match.
- The string ***matches*** the regular expression if it contains the substring.
- Once mastered, regular expressions provide developers with the ability to locate patterns of text in source code and documentation at design time.
- You can also apply regular expressions to text that is subject to algorithmic processing at runtime such as content in HTTP requests or event messages.

Power of Regex is more!

- Regex usage in the real world can get much more complex, and powerful.
  - **Example:** Imagine you need to code to verify contents in the body of an HTTP POST request is free of script injection attacks.
- Injected script code will always appear between <script></script> HTML tags.
- You can apply the regular expression <script>**.***<\/script>, which matches any block of code text bracketed by <script> tags, to the HTTP request body as part of your search for script injection code.

Types of regular expressions

- A regular expression is implemented using a *regular expression engine.*
- A regular expression engine is the underlying software that interprets regular expression patterns and uses those patterns to match text.
- The Linux world has two popular regular expression engines:
  - The POSIX Basic Regular Expression (BRE) engine
  - The POSIX Extended Regular Expression (ERE) engine
- Most Linux utilities at a minimum conform to the POSIX BRE engine specifications, recognizing all the pattern symbols it defines.
- The POSIX ERE engine is often found in programming languages that rely on regular expressions for text filtering.




```
UNIX Tools rocks.
```

match

```
UNIX Tools sucks.
```

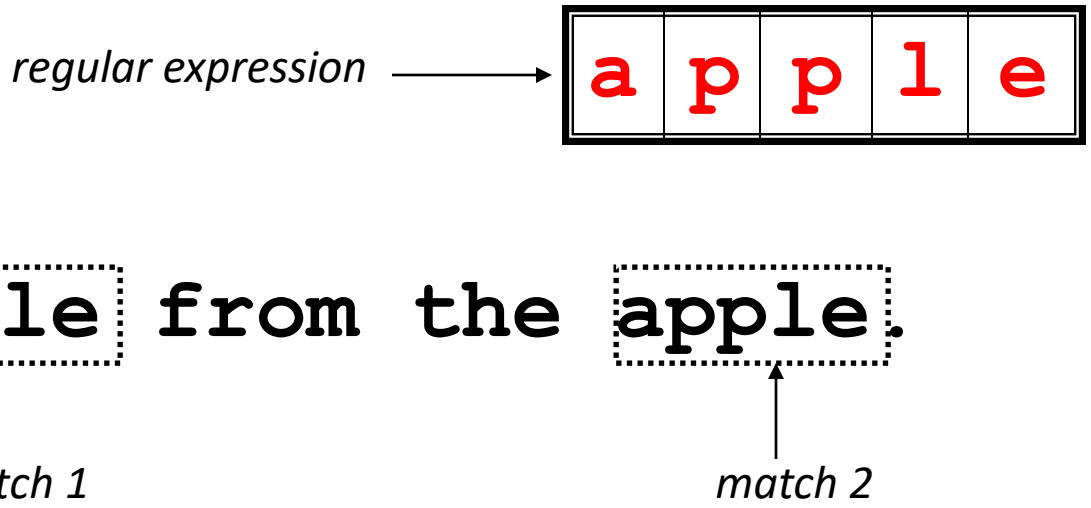
match

```
UNIX Tools is okay.
```

no match

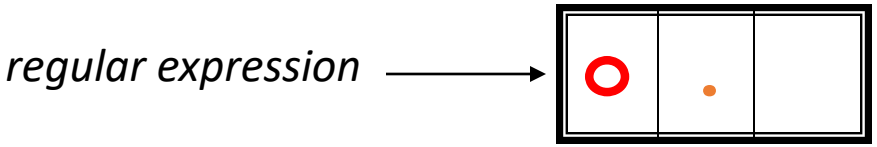
Regular Expressions

- A regular expression can match a string in more than one place.



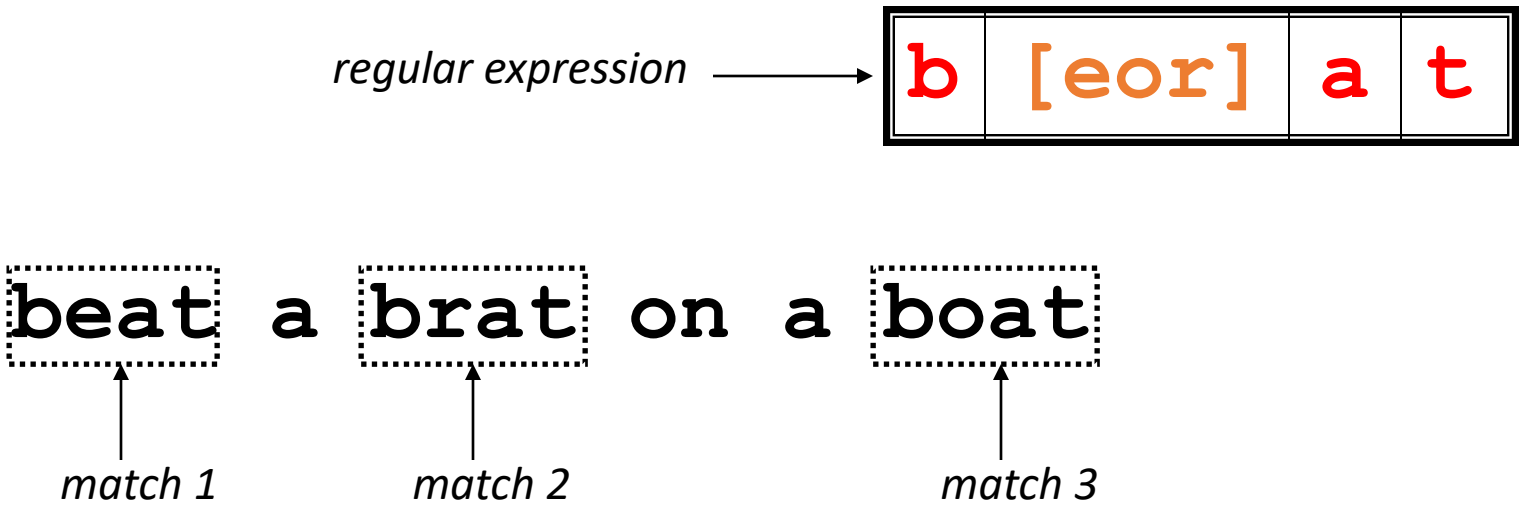
Regular Expressions

- The . regular expression can be used to match any character.



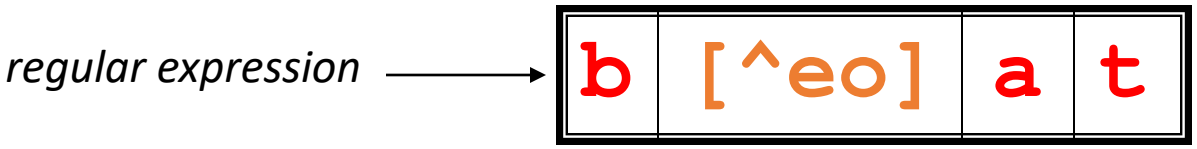
Character Classes

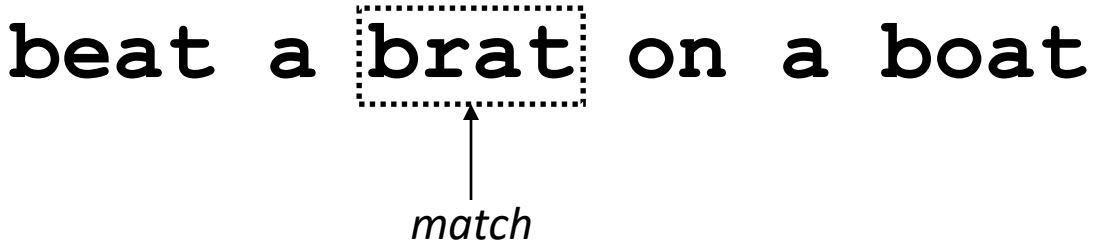
- Character classes `[]` can be used to match any specific set of characters.



Negated Character Classes

- Character classes can be negated with the `[^]` syntax.





More About Character Classes

- `[aeiou]` will match any of the characters `a`, `e`, `i`, `o`, or `u`
- `[kK]orn` will match `korn` or `Korn`

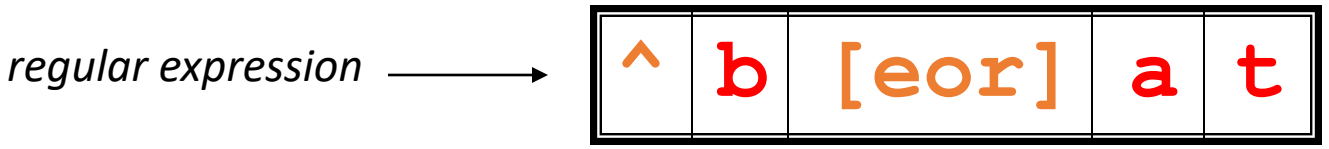
- Ranges can also be specified in character classes
  - `[1-9]` is the same as `[123456789]`
  - `[abcde]` is equivalent to `[a-e]`
  - You can also combine multiple ranges
    - `[abcde123456789]` is equivalent to `[a-e1-9]`
  - Note that the `-` character has a special meaning in a character class ***but only*** if it is used within a range,
    `[-123]` would match the characters `-`, `1`, `2`, or `3`

Named Character Classes

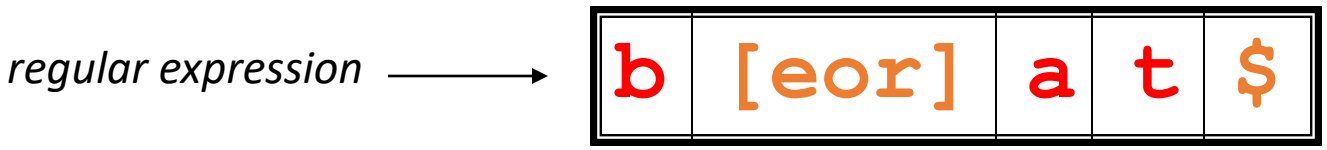
- Commonly used character classes can be referred to by name (*alpha*, *lower*, *upper*, *alnum*, *digit*, *punct*, *cntrl*)
- Syntax `[:name:]`
  - `[a-zA-Z]` `[[:alpha:]]`
  - `[a-zA-Z0-9]` `[[:alnum:]]`
  - `[45a-z]` `[45[:lower:]]`
- Important for portability across languages

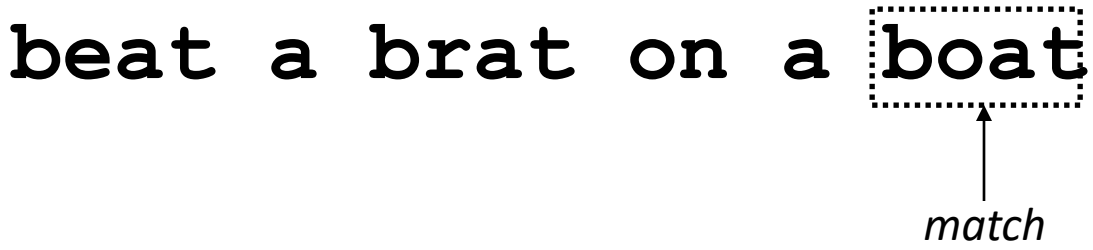
Anchors

- Anchors are used to match at the beginning or end of a line (or both).
- ^ means beginning of the line
- $ means end of the line


regular expression
^ b [eor] a t


beat a brat on a boat
match


regular expression
b [eor] a t $

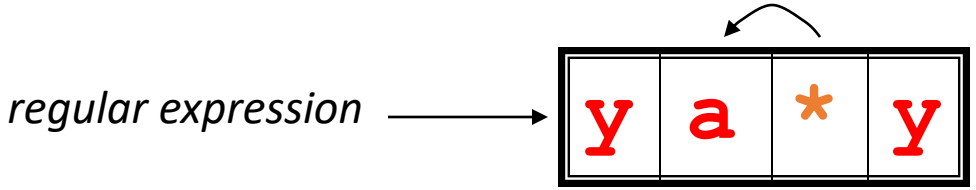

beat a brat on a boat
match

---

^word$ ^$

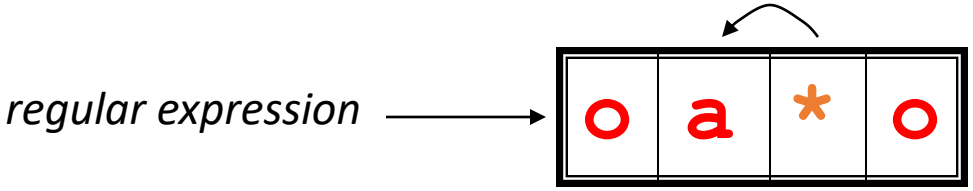
Repetition

- The * is used to define **zero or more** occurrences of the *single* regular expression preceding it.



I got mail, yaaaaaaaaaay!

match



For me to poop on.

match

.*

Repetition Ranges

- Ranges can also be specified
  - `{n,m}` notation can specify a range of repetitions for the immediately preceding regex
  - `{n}` means exactly *n* occurrences
  - `{n,}` means at least *n* occurrences
  - `{n,m}` means at least *n* occurrences but no more than *m* occurrences
- Example:
  - `.{0,}` same as `.*`
  - `a{2,}` same as `aaa*`

Subexpressions

- If you want to group part of an expression so that `*` applies to more than just the previous character, use `(  )` notation
- Subexpresssions are treated like a single character
  - `a*` matches 0 or more occurrences of `a`
  - `abc*` matches `ab`, `abc`, `abcc`, `abccc`, …
  - `(abc)*` matches `abc`, `abcabc`, `abcabcabc`, …
  - `(abc){2,3}` matches `abcabc` or `abcabcabc`

Regular Expression Metacharacters Summary

Metacharacters	Description
* (Asterisk)	This matches zero or more occurrences of the previous character
+ (Plus)	This matches one or more occurrences of the previous character
?	This matches zero or one occurrence of the previous element
. (Dot)	This matches any one character
^	This matches the start of the line
$	This matches the end of line
[...]	This matches any one character within a square bracket
[^...]	This matches any one character that is not within a square bracket
| (Bar)	This matches either the left side or the right side element of |
\{X\}	This matches exactly X occurrences of the previous element
\{X,\}	This matches X or more occurrences of the previous element
\{X,Y\}	This matches X to Y occurrences of the previous element
\(...\)	This groups all the elements
\<	This matches the empty string at the beginning of a word
\>	This matches the empty string at the end of a word
\	This disables the special meaning of the next character

A Website to validate your Regular Expression

- https://regexr.com/

Example1: Regular Expression in a Script

A script to extract the ***domain name*** from a given URL

```
#!/bin/bash
url=$1
regex_pattern="^https?://([^/]+)"
if [[ "$url"  =~ $ regex_pattern ]]; then
        domain=${BASH_REMATCH[1]}
        echo "Domain name: $domain"
else
        echo "Invalid URL"
fi
```

Example2: Regular Expression in a Script

- Script to check if a string is a valid email address

```
#!/bin/bash
email=$1
regex_pattern=" ^[[:alnum:]]+@[[:alnum:]]+\.[[:alnum:]]+$ "
if [[ "$email"  =~ $ regex_pattern ]]; then
        domain=${BASH_REMATCH[1]}
        echo “The email address $domain is a valid address"
else
        echo "Invalid email"
fi
```

Example3: Regular Expression in a Script

- Combining valid months, days, and years regex to form valid dates in MM-DD-YYYY format:

(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[0-1])-(19[7-9][0-9]|20[0-9]{2})

grep

- grep comes from the **ed** (Unix text editor) search command **“global regular expression print”** or g/re/p
- This was such a useful command that it was written as a standalone utility
- There are two other variants, ***egrep*** and ***fgrep*** that comprise the *grep* family
- *grep* is the answer to the moments where you know you want the file that contains a ***specific phrase***, but you can’t remember its name

Family Differences

- **grep -** uses regular expressions for pattern matching
- **fgrep -** file grep, does not use regular expressions, only matches fixed strings but can get search strings from a file
- **egrep -** extended grep, uses a more powerful set of regular expressions but does not support backreferencing, generally the fastest member of the grep family

Syntax

- Regular expression concepts we have seen so far are common to **grep** and **egrep**.
- grep and egrep have different syntax
  - **grep**: BREs
  - **egrep**: EREs
- Major syntax differences:
  - **grep**: \(and \), \{ and \}
  - **egrep**: (and), { and }

Protecting Regex Metacharacters

- Since many of the special characters used in regex also have special meaning to the shell, ***it's a good idea to get in the habit of single quoting your regex***
  - This will protect any special characters from being operated on by the shell
  - If you habitually, do it, you won't have to worry about when it is necessary

Escaping Special Characters

- Even though we are single quoting our ***regex*** so the shell won't interpret the special characters, sometimes we still want to use an operator as itself
- To do this, we "escape" the character with a \ (backslash)

- Suppose we want to search for the character sequence 'a*b*'
  - Unless we do something special, this will match zero or more 'a's followed by zero or more 'b's, *not what we want*
  - **'a*b*'** will fix this - now the asterisks are treated as regular characters

Egrep: Alternation

- Regex also provides an alternation character | for matching one or another subexpression
  - **(T|Fl)an** will match ‘Tan’ or ‘Flan’
  - **^(From|Subject):** will match the From and Subject lines of a typical email message
    - It matches a beginning of line followed by either the characters ‘From’ or ‘Subject’ followed by a ‘:’
- Subexpressions are used to limit the scope of the alternation
  - **At(ten|nine)tion** then matches “Attention” or “Atninetion”, not “Atten” or “ninetion” as would happen without the parenthesis - **Atten|ninetion**

Egrep: Repetition Shorthands

- The * (star) has already been seen to specify zero or more occurrences of the immediately preceding character

- + (plus) means “one or more”
  - **abc+d** will match ‘abcd’, ‘abccd’, or ‘abccccccd’ but will not match ‘abd’
  - Equivalent to **{1,}**

Egrep: Repetition Shorthands cont

- The '**?**' (question mark) specifies an optional character, the single character that immediately precedes it
  - **July?** will match 'Jul' or 'July'
  - Equivalent to **{0,1}**
  - Also equivalent to **(Jul|July)**
- The *****, **?**, and **+** are known as *quantifiers* because they specify the quantity of a match
- Quantifiers can also be used with subexpressions
  - **(a*c)+** will match 'c', 'ac', 'aac' or 'aacaacac' but will not match 'a' or a blank line

Grep: Backreferences

- Sometimes it is handy to be able to refer to a match that was made earlier in a regex
- This is done using *backreferences*
  - ***n*** is the backreference specifier, where *n* is a number
- For example, ***to find if the first word of a line is the same as the last:***
  - **^\([[:alpha:]]\{1,\}\).*\1$**
  - The **\([[:alpha:]]\{1,\}\)** matches 1 or more letters

Practical Regex Examples

- Variable names in C
  - **[a-zA-Z_][a-zA-Z_0-9]***
- Dollar amount with optional cents
  - **\$[0-9]+(\.[0-9][0-9])?**
- Time of day
  - **(1[012]|[1-9]):[0-5][0-9] (am|pm)**
- HTML headers <h1> <H1> <h2> …
  - **<[hH][1-4]>**

grep Family

- Syntax

  grep [-hilnv] [-e expression] [filename]

  egrep [-hilnv] [-e expression] [-f filename] [expression] [filename]

  fgrep [-hilnxv] [-e string] [-f filename] [string] [filename]

  - **-h** Do not display filenames
  - **-i** Ignore case
  - **-l** List only filenames containing matching lines
  - **-n** Precede each matching line with its line number
  - **-v** Negate matches
  - **-x** Match whole line only (*fgrep* only)
  - **-e** *expression* Specify expression as option
  - **-f** *filename* Take the regular expression (egrep) or a list of strings (fgrep) from *filename*

Fun with the Dictionary

- **/usr/dict/words** contains about 25,000 words
  - **egrep hh /usr/dict/words**
    - beachhead
    - highhanded
    - withheld
    - withhold
- **egrep** as a simple spelling checker: Specify plausible alternatives you know

  egrep "n(ie|ei)ther" /usr/dict/words

  neither
- How many words have 3 a's one letter apart?
  - **egrep a.a.a /usr/dict/words | wc –l**
    - 54
  - **egrep u.u.u /usr/dict/words**
    - cumulus

BENNETT UNIVERSITY
THE TIMES GROUP

Thanks

Q & A