Unix Programming

Regular Expressions

Nguyen Thanh Hung
Software Engineering Department
Hanoi University of Science and Technology

What Is a Regular Expression?

- **A regular expression (*regex*) describes a set of possible input strings.**
- ***Regular expressions* descend from a fundamental concept in Computer Science called *finite automata* theory**
- ***Regular expressions* are endemic to Unix**
  - **vi**, **ed**, **sed**, and **emacs**
  - **awk**, **tcl**, **perl** and **Python**
  - **grep**, **egrep**, **fgrep**
  - **compilers**

Regular Expressions

- **The simplest regular expressions are a string of literal characters to match.**
- **The string *matches* the regular expression if it contains the substring.**

regular expression → `c k s`

`UNIX Tools rocks.`

match

`UNIX Tools sucks.`

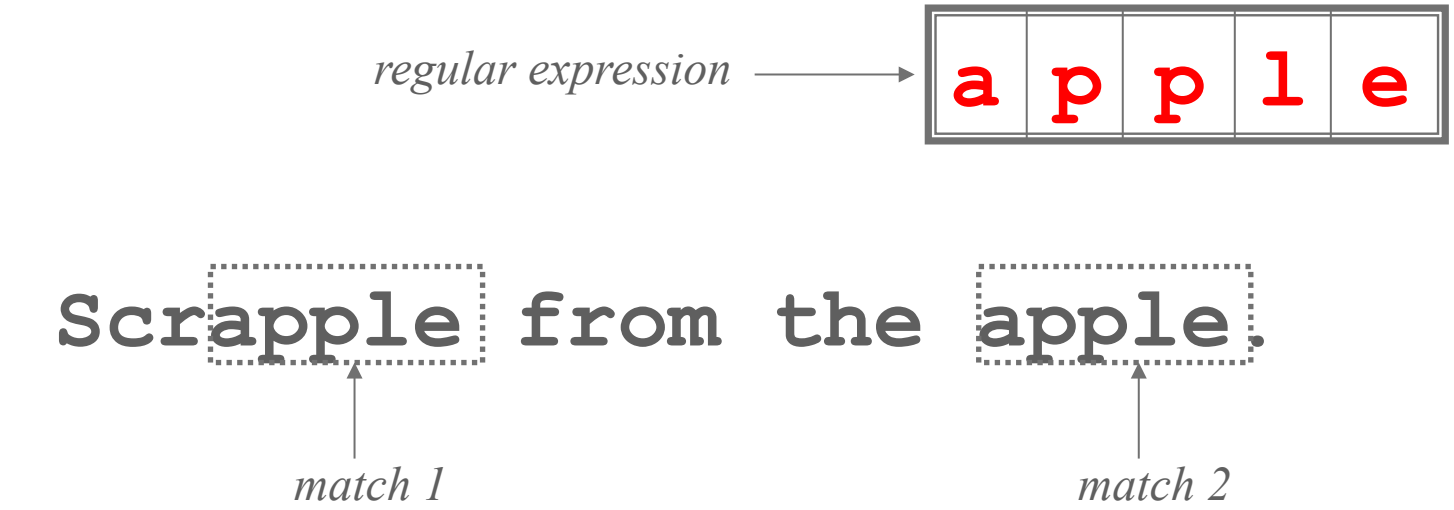
match

`UNIX Tools is okay.`

no match

Regular Expressions

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

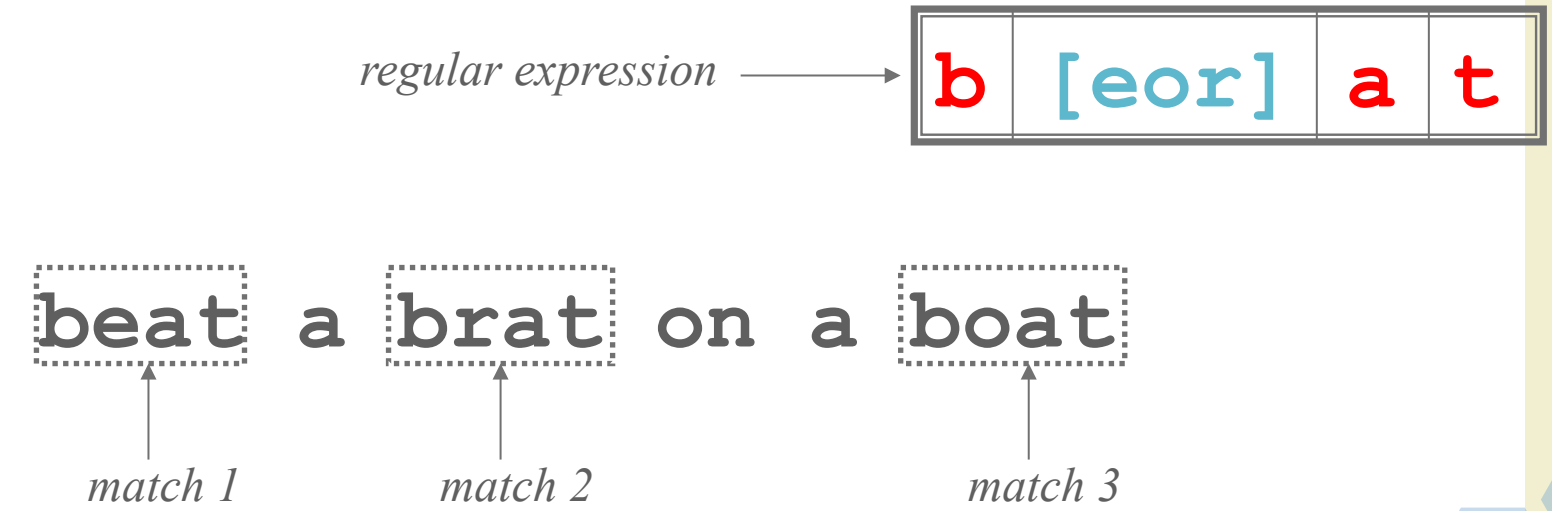


Regular Expressions

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

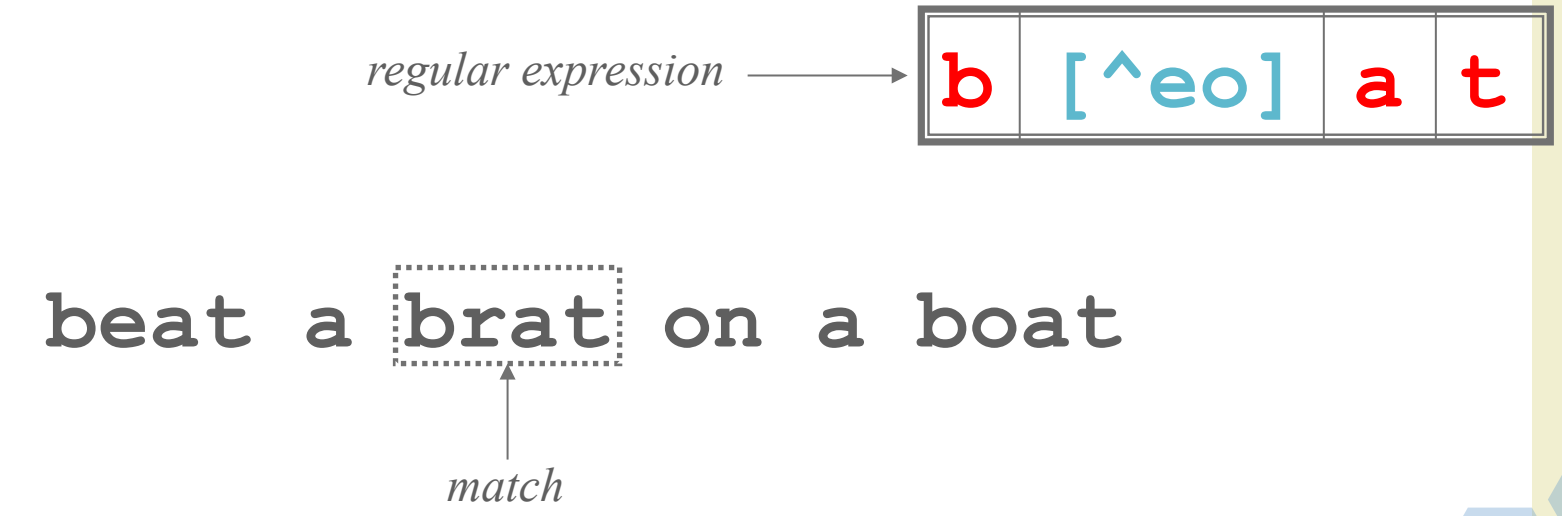
Character Classes

- **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

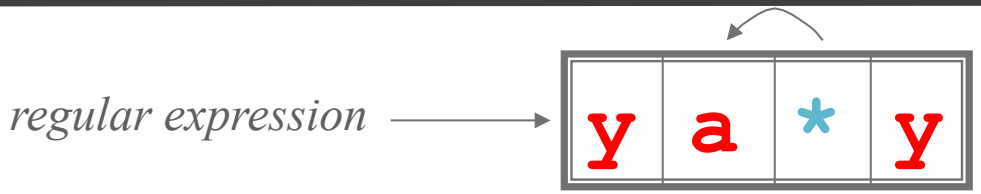
match

---

^word$ ^$

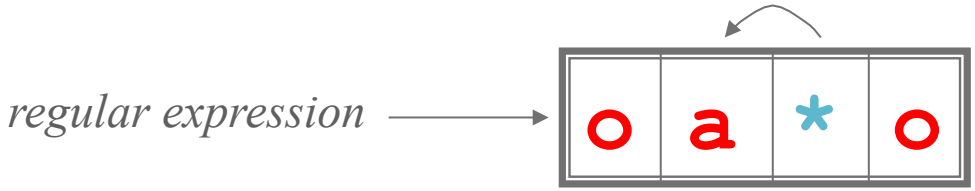
Repetition

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



I got mail, yaaaaaaaaaaay!

match



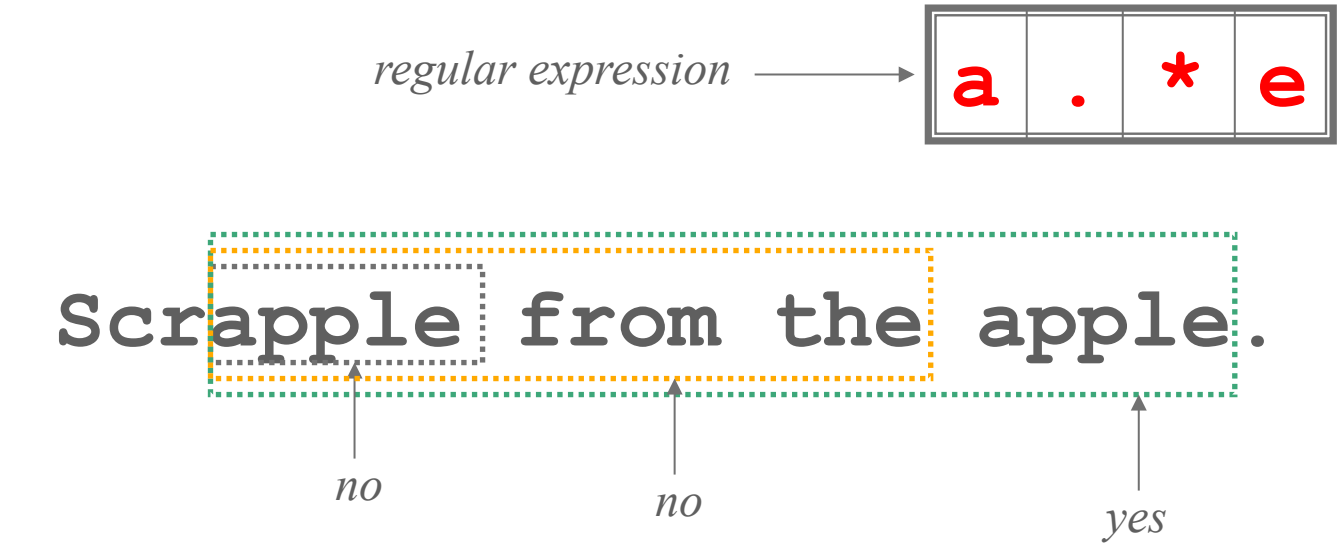
For me to poop on.

match

.*

Match length

- **A match will be the longest string that satisfies the regular expression.**



Repetition Ranges

- **Ranges can also be specified**
  - `{ }` 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 `*` or `{ }` 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`