Unicode Search of Dirty Data, Or: How I Learned to Stop Worrying and Love Unicode Technical Standard #18

By

Jon Stewart and Joel Uckelman

Presented At

The Digital Forensic Research Conference

DFRWS 2013 USA Monterey, CA (Aug 4th - 7th)

DFRWS is dedicated to the sharing of knowledge and ideas about digital forensics research. Ever since it organized the first open workshop devoted to digital forensics in 2001, DFRWS continues to bring academics and practitioners together in an informal environment. As a non-profit, volunteer organization, DFRWS sponsors technical working groups, annual conferences and challenges to help drive the direction of research and development.

http:/dfrws.org

Unicode Search of Dirty Data, Or: How I Learned to Stop Worrying and Love Unicode Technical Standard #18

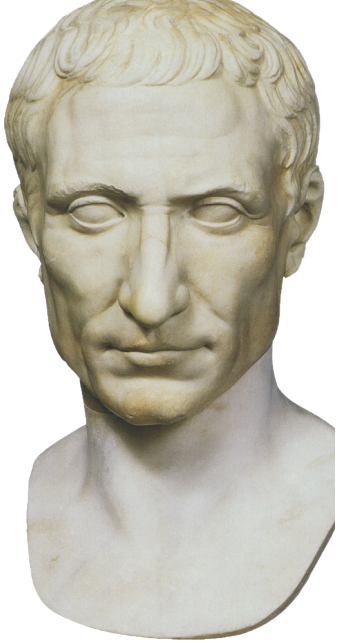
Jon Stewart Joel Uckelman*

Lightbox Technologies
Arlington, VA
{jon,joel}@lightboxtechnologies.com

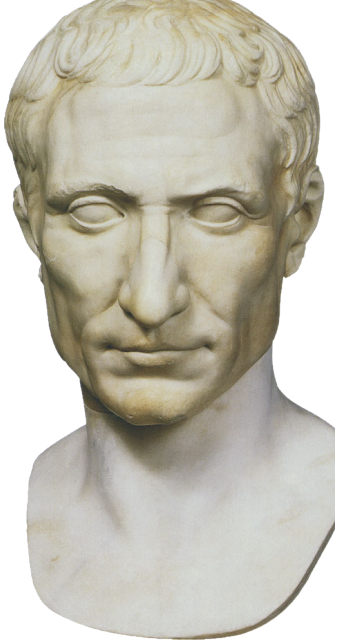
DFRWS 2013

A Motivating Example

A Motivating Example

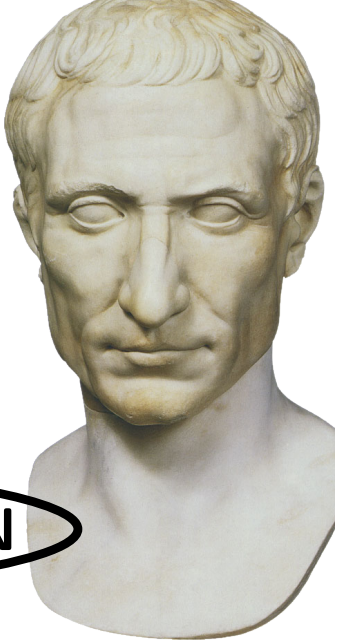


A Motivating Example



A Motivating Example

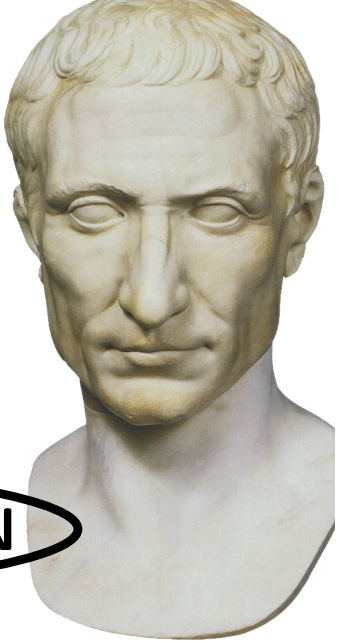
A Motivating Example



A T T A C K A T D A W N

41	54	54	41	43	4B	20	41	54	20	44	41	57	4E

A Motivating Example

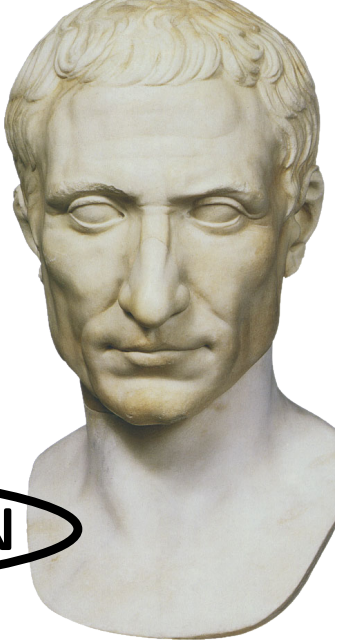


A T T A C K A T D A W N

latin1

41	54	54	41	43	4B	20	41	54	20	44	41	57	4E

A Motivating Example



A T T A C K A T D A W N

latin1

41	54	54	41	43	4B	20	41	54	20	44	41	57	4E

D3	63	63	D3	51	9D	4C	D3	63	4C	45	D3	4B	09

A Motivating Example

A T T A C K A T D A W N

latin1

41	54	54	41	43	4B	20	41	54	20	44	41	57	4E

OCE(latin1)

D3	63	63	D3	51	9D	4C	D3	63	4C	45	D3	4B	09

A Motivating Example

A T T A C K A T D A W N

latin1

41	54	54	41	43	4B	20	41	54	20	44	41	57	4E

OCE(latin1)

D3	63	63	D3	51	9D	4C	D3	63	4C	45	D3	4B	09

41	00	54	00	54	00	41	00	43	00	4B	00	20	00	41	00	54	00	20	00	44	00	41	00	57	00	4E	00

A Motivating Example

A T T A C K A T D A W N

latin1

41	54	54	41	43	4B	20	41	54	20	44	41	57	4E

OCE(latin1)

D3	63	63	D3	51	9D	4C	D3	63	4C	45	D3	4B	09

41	00	54	00	54	00	41	00	43	00	4B	00	20	00	41	00	54	00	20	00	44	00	41	00	57	00	4E	00

UTF-16LE

A Motivating Example

A T T A C K A T D A W N

latin1

41	54	54	41	43	4B	20	41	54	20	44	41	57	4E

OCE(latin1)

D3	63	63	D3	51	9D	4C	D3	63	4C	45	D3	4B	09

41	00	54	00	54	00	41	00	43	00	4B	00	20	00	41	00	54	00	20	00	44	00	41	00	57	00	4E	00

UTF-16LE

00	41	00	54	00	54	00	41	00	43	00	4B	00	20	00	41	00	54	00	20	00	44	00	41	00	57	00	4E

A Motivating Example

A T T A C K A T D A W N

latin1

41	54	54	41	43	4B	20	41	54	20	44	41	57	4E

OCE(latin1)

D3	63	63	D3	51	9D	4C	D3	63	4C	45	D3	4B	09

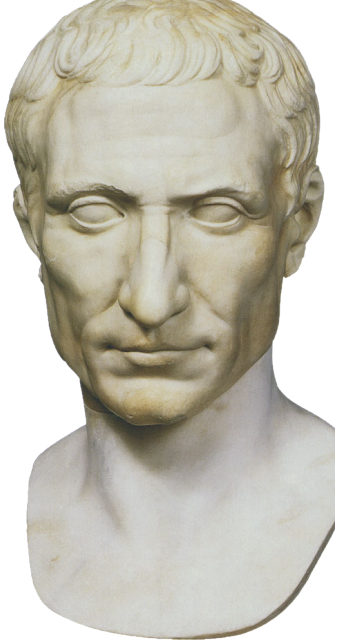
41	00	54	00	54	00	41	00	43	00	4B	00	20	00	41	00	54	00	20	00	44	00	41	00	57	00	4E	00

UTF-16LE

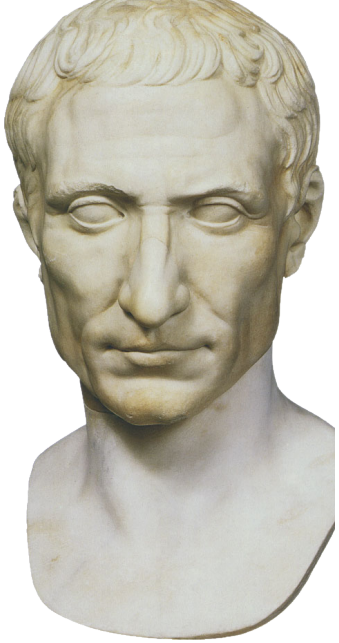
00	41	00	54	00	54	00	41	00	43	00	4B	00	20	00	41	00	54	00	20	00	44	00	41	00	57	00	4E

UTF-16BE

A Motivating Example

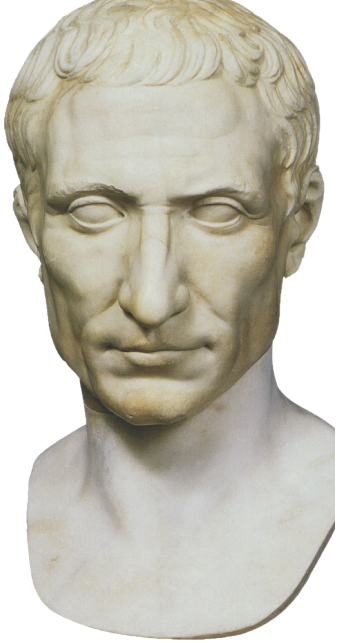


A Motivating Example



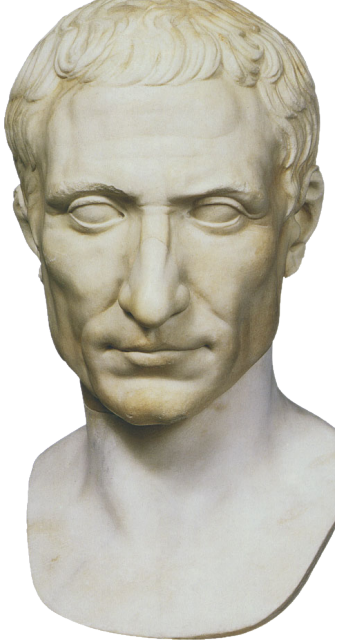
- Many possible encodings for text, all different

A Motivating Example



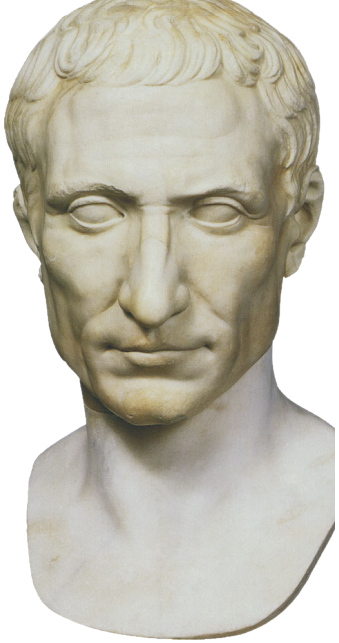
- Many possible encodings for text, all different
- We don't want to search serially for each encoding

A Motivating Example



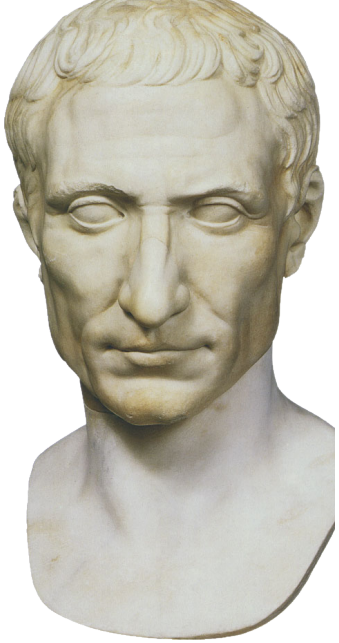
- Many possible encodings for text, all different
- We don't want to search serially for each encoding
- We'd like to have encoding-independent patterns

A Motivating Example



- Many possible encodings for text, all different
- We don't want to search serially for each encoding
- We'd like to have encoding-independent patterns
- We have a multipattern search tool (Lightgrep)

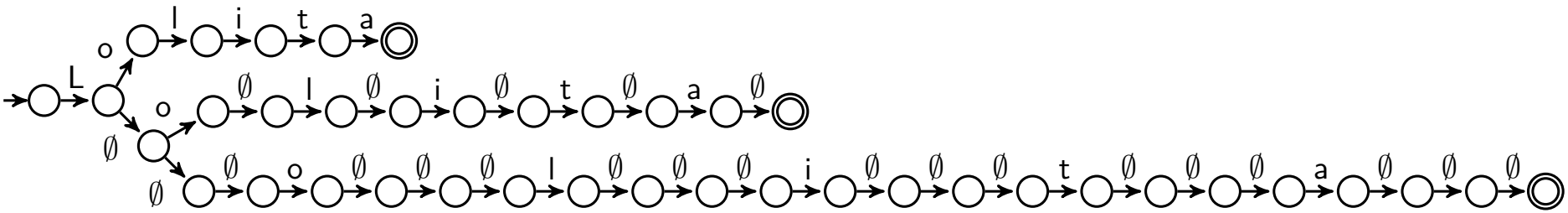
A Motivating Example



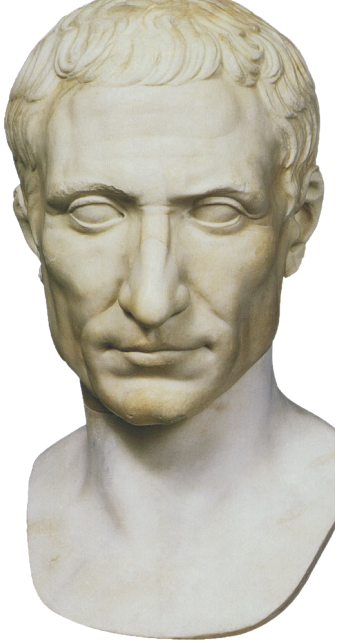
- Many possible encodings for text, all different
- We don't want to search serially for each encoding
- We'd like to have encoding-independent patterns
- We have a multipattern search tool (Lightgrep)

What to do?

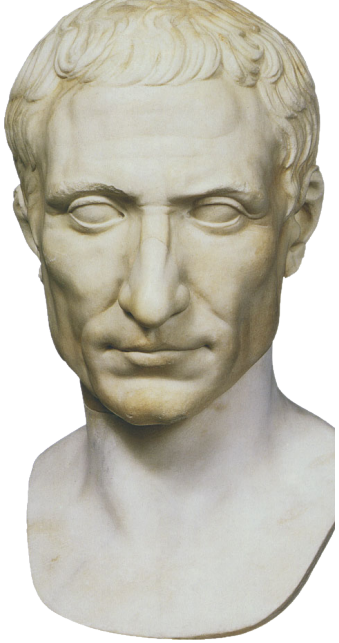
Multiencoding search is multipattern search.

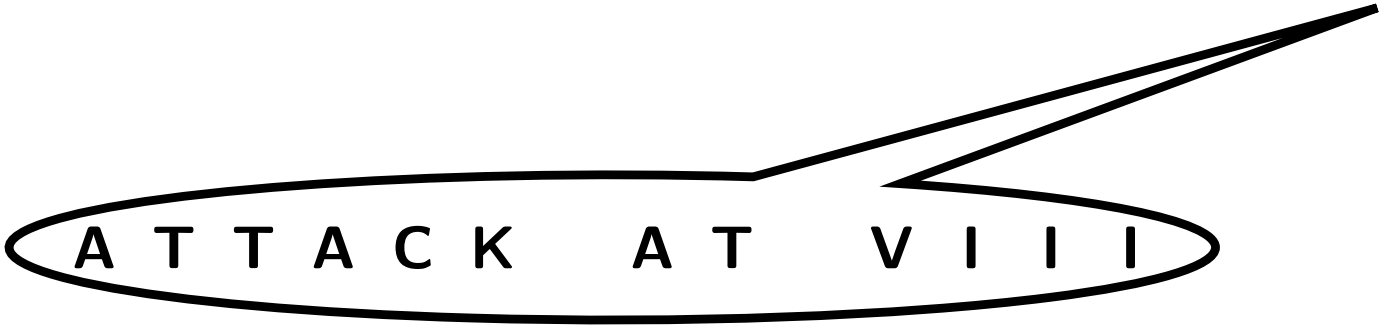


A Motivating Example

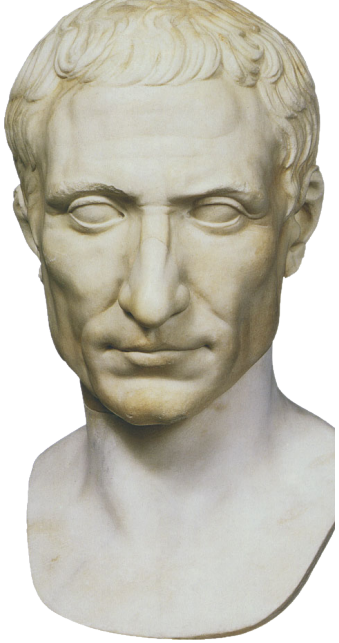


A Motivating Example





A Motivating Example



A T T A C K A T I V

A T T A C K A T V

A T T A C K A T V I

A T T A C K A T V I I

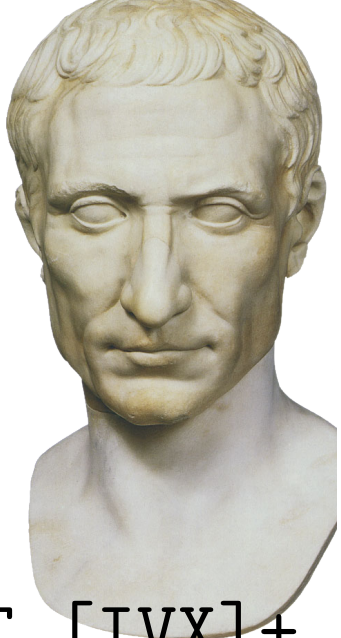
A T T A C K A T V I I I

A T T A C K A T I X

A T T A C K A T X

A T T A C K A T X I

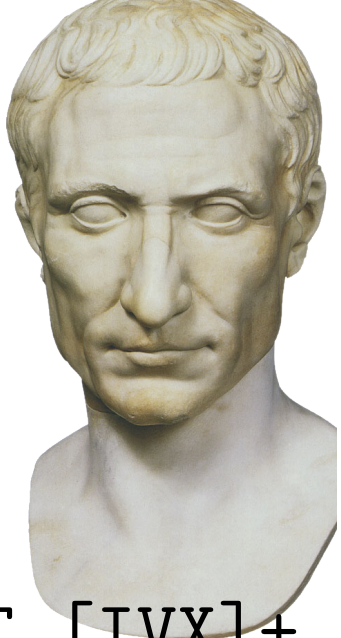
A Motivating Example



A T T A C K A T I V
A T T A C K A T V
A T T A C K A T V I
A T T A C K A T V I I
A T T A C K A T V I I I
A T T A C K A T I X
A T T A C K A T X
A T T A C K A T X I

ATTACK AT [IVX]+

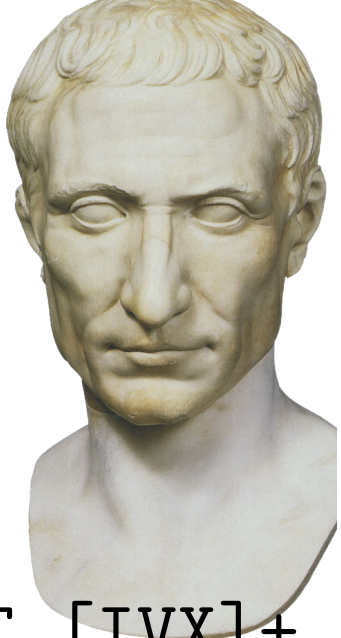
A Motivating Example



A T T A C K A T I V
A T T A C K A T V
A T T A C K A T V I
A T T A C K A T V I I
A T T A C K A T V I I I
A T T A C K A T I X
A T T A C K A T X
A T T A C K A T X I

ATTACK AT [IVX]+

A Motivating Example



A T T A C K A T I V
A T T A C K A T V
A T T A C K A T V I
A T T A C K A T V I I
A T T A C K A T V I I I
A T T A C K A T I X
A T T A C K A T X
A T T A C K A T X I

ATTACK AT [IVX]+

VIII ROMAN NUMERAL EIGHT (U+2167)

A Motivating Example

A T T A C K A T I V
A T T A C K A T V
A T T A C K A T V I
A T T A C K A T V I I
A T T A C K A T V I I I
A T T A C K A T I X
A T T A C K A T X
A T T A C K A T X I

ATTACK AT [IVX]+

VIII ROMAN NUMERAL EIGHT (U+2167)

A Motivating Example

A T T A C K A T I V
A T T A C K A T V
A T T A C K A T V I
A T T A C K A T V I I
A T T A C K A T X
A T T A C K A T X
A T T A C K A T X I

ATTACK AT [IVX]+

GORY DETAILS

VIII ROMAN NUMERAL EIGHT (U+2167)

What is Unicode?

Before Unicode:

- Hundreds of character encodings

What is Unicode?

Before Unicode:

- Hundreds of character encodings
- Character set = character encoding

What is Unicode?

Before Unicode:

- Hundreds of character encodings
- Character set = character encoding
- Small, often single-byte, character sets ($\leq$ 256 characters)

What is Unicode?

Before Unicode:

- Hundreds of character encodings
- Character set = character encoding
- Small, often single-byte, character sets ($\leq$ 256 characters)
- Interoperability was messy

What is Unicode?

Before Unicode:

- Hundreds of character encodings
- Character set = character encoding
- Small, often single-byte, character sets ($\leq$ 256 characters)
- Interoperability was messy
- Multilanguage documents were difficult

What is Unicode?

Before Unicode:

- Hundreds of character encodings
- Character set = character encoding
- Small, often single-byte, character sets ($\leq$ 256 characters)
- Interoperability was messy
- Multilanguage documents were difficult

Unicode is an attempt to represent all characters used by humans.

What is Unicode?

Before Unicode:

- Hundreds of character encodings
- Character set = character encoding
- Small, often single-byte, character sets ($\leq$ 256 characters)
- Interoperability was messy
- Multilanguage documents were difficult

Unicode is an attempt to represent all characters used by humans.

After Unicode:

What is Unicode?

Before Unicode:

- Hundreds of character encodings
- Character set = character encoding
- Small, often single-byte, character sets ($\leq$ 256 characters)
- Interoperability was messy
- Multilanguage documents were difficult

Unicode is an attempt to represent all characters used by humans.

After Unicode:

- Only a handful of actively-used encodings

What is Unicode?

Before Unicode:

- Hundreds of character encodings
- Character set = character encoding
- Small, often single-byte, character sets ($\leq$ 256 characters)
- Interoperability was messy
- Multilanguage documents were difficult

Unicode is an attempt to represent all characters used by humans.

After Unicode:

- Only a handful of actively-used encodings
- Character set $\neq$ character encoding

What is Unicode?

Before Unicode:

- Hundreds of character encodings
- Character set = character encoding
- Small, often single-byte, character sets ($\leq$ 256 characters)
- Interoperability was messy
- Multilanguage documents were difficult

Unicode is an attempt to represent all characters used by humans.

After Unicode:

- Only a handful of actively-used encodings
- Character set $\neq$ character encoding
- Unicode is a character set.
  UTF-8, UTF-16LE, etc. are character encodings for Unicode.

What is Unicode?

Before Unicode:

- Hundreds of character encodings
- Character set = character encoding
- Small, often single-byte, character sets ($\leq$ 256 characters)
- Interoperability was messy
- Multilanguage documents were difficult

Unicode is an attempt to represent all characters used by humans.

After Unicode:

- Only a handful of actively-used encodings
- Character set $\neq$ character encoding
- Unicode is a character set.
  UTF-8, UTF-16LE, etc. are character encodings for Unicode.
- Huge character set—space for 1.1m *code points*

Unicode Terminology

Code point A (possibly named) character, identified by a hex number

Unicode Terminology

Code point A (possibly named) character, identified by a hex number

E.g.: U+0058, LATIN CAPITAL LETTER X

Unicode Terminology

Code point A (possibly named) character, identified by a hex number

E.g.: U+0058, LATIN CAPITAL LETTER X

Code points range from U+0000 to U+10FFFF

Unicode and Regular Expressions

- Unicode contains many weird and wonderful things.

ð ऌ Д € 獼 Φ ﷺ Ꮹ ⁙ ∰ 💩 ☎

- Unicode contains many weird and wonderful things.

ð ऌ Д € 獼 Φ ﷺ Ꭽ ⁙ ∰ 💩 ☎

- Regular expressions were designed for ASCII text.

Unicode and Regular Expressions

- Unicode contains many weird and wonderful things.

ð ऌ Д € 獼 Φ ﷺ [illegible] ⁙ ∰ 💩 ☎

- Regular expressions were designed for ASCII text.

- Unicode Technical Standard #18 gives guidelines for supporting Unicode in regular expressions:

Unicode and Regular Expressions

- Unicode contains many weird and wonderful things.

ð ऌ Д € 獮 Φ ﷺ դ ⡵ ∰ 💩 ☎

- Regular expressions were designed for ASCII text.

- Unicode Technical Standard #18 gives guidelines for supporting Unicode in regular expressions:
    - Level 1: Basic Unicode Support

- Unicode contains many weird and wonderful things.

ð ऌ Д € 獮 Φ ﷺ Ꮿ ⁙ ∰ 💩 ☎

- Regular expressions were designed for ASCII text.
- Unicode Technical Standard #18 gives guidelines for supporting Unicode in regular expressions:
  - Level 1: Basic Unicode Support
  - Level 2: Extended Unicode Support

Unicode and Regular Expressions

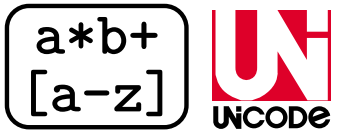
- Unicode contains many weird and wonderful things.

ð ಠ_ಠ Д € 獼 Φ ﷺ ᘐ :: ∯ 💩 ☎

- Regular expressions were designed for ASCII text.

- Unicode Technical Standard #18 gives guidelines for supporting Unicode in regular expressions:
  - Level 1: Basic Unicode Support
  - Level 2: Extended Unicode Support
  - Level 3: Tailored Support

What is that? How do I type it?



- Some characters are indistinguishable from one another.

What is that? How do I type it?

- Some characters are indistinguishable from one another.

  K LATIN CAPITAL LETTER K (U+004B)
  K KELVIN SIGN (U+212A)

What is that? How do I type it?

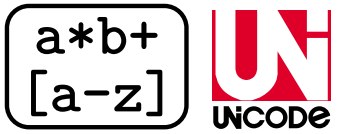
- Some characters are indistinguishable from one another.

  K LATIN CAPITAL LETTER K (U+004B)

  K KELVIN SIGN (U+212A)

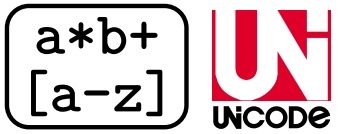
tab (U+0009)
SPACE (U+0020)
MONGOLIAN VOWEL SEPARATOR (U+180E)
EN QUAD (U+2000)
EM QUAD (U+2001)
EN SPACE (U+2002)
EM SPACE (U+2003)
THREE-PER-EM SPACE (U+2004)
FOUR-PER-EM SPACE (U+2005)
SIX-PER-EM SPACE (U+2006)
FIGURE SPACE (U+2007)
PUNCTUATION SPACE (U+2008)
THIN SPACE (U+2009)
HAIR SPACE (U+200A)
NARROW NO-BREAK SPACE (U+202F)
MEDIUM MATHEMATICAL SPACE (U+205F)
IDEOGRAPHIC SPACE (U+3000)

What is that? How do I type it?



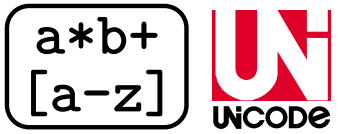
- Some characters are indistinguishable from one another.
- Some characters will be unfamiliar to you.

What is that? How do I type it?



- Some characters are indistinguishable from one another.
- Some characters will be unfamiliar to you.
- Some characters you won't know how to type.

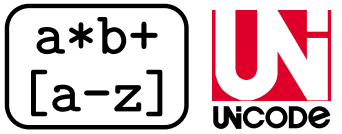
What is that? How do I type it?



- Some characters are indistinguishable from one another.
- Some characters will be unfamiliar to you.
- Some characters you won't know how to type.

UTS #18 requires:

What is that? How do I type it?



- Some characters are indistinguishable from one another.
- Some characters will be unfamiliar to you.
- Some characters you won't know how to type.

UTS #18 requires:

- Code points can be specified by hex notation (RL1.2)

$$\texttt{\textbackslash x\{10DA\}} = \mathfrak{m}$$

What is that? How do I type it?

- Some characters are indistinguishable from one another.
- Some characters will be unfamiliar to you.
- Some characters you won't know how to type.

UTS #18 requires:

- Code points can be specified by hex notation (RL1.2)

  `\x{10DA}` = ϚϚ

- Code points can be specified by name (RL2.5)

  `\N{latin small letter sharp s}` = ß

What is that? How do I type it?

- Some characters are indistinguishable from one another.
- Some characters will be unfamiliar to you.
- Some characters you won't know how to type.

UTS #18 requires:

- Code points can be specified by hex notation (RL1.2)

  `\x{10DA}` = ϚϚ

- Code points can be specified by name (RL2.5)

  `\N{latin small letter sharp s}` = ß

- All code points are supported (RL1.7)

Adventures in Case Sensitivity

- How many characters match `[a-z]` case insensitively? 52

Adventures in Case Sensitivity

- How many characters match `[a-z]` case insensitively? ~~52~~

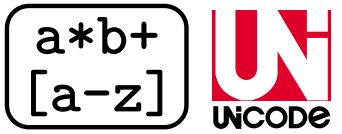
Adventures in Case Sensitivity

- How many characters match [a-z] case insensitively? ~~52~~ 54 !!!

Adventures in Case Sensitivity

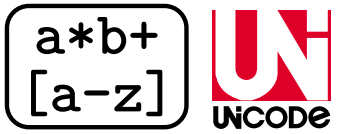
- How many characters match [a-z] case insensitively? ~~52~~ 54 !!!
- What?!

Adventures in Case Sensitivity



- How many characters match [a-z] case insensitively? ~~52~~ 54 !!!
- What?!
  - a–z and A–Z, as expected but also

Adventures in Case Sensitivity



- How many characters match [a-z] case insensitively? ~~52~~ 54 !!!
- What?!
  - a–z and A–Z, as expected but also
  - K KELVIN SIGN (U+212A), and

Adventures in Case Sensitivity

- How many characters match [a-z] case insensitively? ~~52~~ 54 !!!
- What?!
  - a–z and A–Z, as expected but also
  - K KELVIN SIGN (U+212A), and
  - ſ LATIN SMALL LETTER LONG S (U+017F)

Adventures in Case Sensitivity

- How many characters match [a-z] case insensitively? ~~52~~ 54 !!!
- What?!
  - a–z and A–Z, as expected but also
  - K KELVIN SIGN (U+212A), and
  - ſ LATIN SMALL LETTER LONG S (U+017F)
- UTS #18 requires proper handling of case-insensitivity (RL1.5)

Adventures in Case Sensitivity

- How many characters match `[a-z]` case insensitively? ~~52~~ 54 !!!
- What?!
  - a–z and A–Z, as expected but also
  - K KELVIN SIGN (U+212A), and
  - ſ LATIN SMALL LETTER LONG S (U+017F)
- UTS #18 requires proper handling of case-insensitivity (RL1.5)
- E.g., `happiness` case-insensitively matches the last word in "the pursuit of Happineſs"

Adventures in Case Sensitivity

- How many characters match `[a-z]` case insensitively? ~~52~~ 54 !!!
- What?!
  - a–z and A–Z, as expected but also
  - K KELVIN SIGN (U+212A), and
  - ſ LATIN SMALL LETTER LONG S (U+017F)
- UTS #18 requires proper handling of case-insensitivity (RL1.5)
- E.g., `happiness` case-insensitively matches the last word in "the pursuit of Happineſs"
- No missed hits when case-insensitively searching unfamiliar scripts

Properties

- *Properties* are named sets of code points. E.g.:

Alphabetic	Letter	Mark	Separator
Uppercase	Number	Symbol	Assigned
Lowercase	Punctuation	Control	...

Properties

- *Properties* are named sets of code points. E.g.:

Alphabetic	Letter	Mark	Separator
Uppercase	Number	Symbol	Assigned
Lowercase	Punctuation	Control	...

- *Scripts* are properties:

Latin	Arabic	Hangul	Hebrew
Greek	Han	Katakana	Devanagari
Cyrillic	Thai	Hiragana	...

Properties

- *Properties* are named sets of code points. E.g.:

Alphabetic	Letter	Mark	Separator
Uppercase	Number	Symbol	Assigned
Lowercase	Punctuation	Control	...

- *Scripts* are properties:

Latin	Arabic	Hangul	Hebrew
Greek	Han	Katakana	Devanagari
Cyrillic	Thai	Hiragana	...

- UTS #18: properties accessible as character classes (RL1.2, RL2.7)

Properties

- *Properties* are named sets of code points. E.g.:

Alphabetic	Letter	Mark	Separator
Uppercase	Number	Symbol	Assigned
Lowercase	Punctuation	Control	...

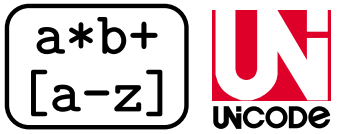
- *Scripts* are properties:

Latin	Arabic	Hangul	Hebrew
Greek	Han	Katakana	Devanagari
Cyrillic	Thai	Hiragana	...

- UTS #18: properties accessible as character classes (RL1.2, RL2.7)

```
\p{Cyrillic}{4,}
[\p{Arabic}\p{Whitespace}\p{Punctuation}]+
```

Set Operations on Character Classes



- How do I match *lowercase* Greek?

Set Operations on Character Classes

- How do I match *lowercase* Greek?

  [α–ω] misses ϊ, ϋ, ό, ύ, ώ

Set Operations on Character Classes

- How do I match *lowercase* Greek?

  [α–ω] misses ϊ, ϋ, ό, ύ, ώ

- UTS #18 requires union, intersection, subtraction for CCs (RL1.3)

Set Operations on Character Classes

- How do I match *lowercase* Greek?

  `[α-ω]` misses ϊ, ϋ, ό, ύ, ώ

- UTS #18 requires union, intersection, subtraction for CCs (RL1.3)

```
[\p{Greek}&&\p{Lowercase}]
```

Set Operations on Character Classes

- How do I match *lowercase* Greek?

  `[α-ω]` misses ϊ, ϋ, ό, ύ, ώ

- UTS #18 requires union, intersection, subtraction for CCs (RL1.3)

```
[\p{Greek}&&\p{Lowercase}]

[\p{ASCII}--\p{Whitespace}]
```

UTS #18 Support

	lightgrep	ICU 50	Perl 5.16	Java 7	Python regex
Level 1: Basic Unicode Support	○	●	○	●	●
RL1.1 Hex Notation	●	●	●	●	●
RL1.2 Properties	●	●	●	●	●
RL1.3 Subtraction and Intersection	●	●	○	●	●
RL1.4 Simple Word Boundaries		●	●	●	●
RL1.5 Simple Loose Matches	●	●	●	●	●
RL1.6 Line Boundaries		●	○	●	●
RL1.7 Supplementary Code Points	●	●	●	●	●
Level 2: Extended Unicode Support	○	○	○	○	○
RL2.1 Canonical Equivalents				●	
RL2.2 Extended Grapheme Clusters		○	○		●
RL2.3 Default Word Boundaries		●			
RL2.4 Default Case Conversion					
RL2.5 Name Properties	●	●	●		●
RL2.6 Wildcards in Property Values					
RL2.7 Full Properties	●		●		●
Level 3: Tailored Support	○		○		
RL3.1 Tailored Punctuation					
RL3.2 Tailored Grapheme Clusters					
RL3.6 Context Matching			○		
RL3.7 Incremental Matches	●				
RL3.9 Possible Match Sets	●				
RL3.11 Submatchers					

● = full support, ○ = partial support

UTS #18 Support

	lightgrep	ICU 50	Perl 5.16	Java 7	Python regex
Level 1: Basic Unicode Support	○	●	○	●	●
RL1.1 Hex Notation	●	●	●	●	●
RL1.2 Properties	●	●	●	●	●
RL1.3 Subtraction and Intersection	●	●	○	●	●
RL1.4 Simple Word Boundaries		●	●	●	●
RL1.5 Simple Loose Matches	●	●	●	●	●
RL1.6 Line Boundaries		●	○	●	●
RL1.7 Supplementary Code Points	●	●	●	●	●
Level 2: Extended Unicode Support	○	○	○	○	○
RL2.1 Canonical Equivalents				●	
RL2.2 Extended Grapheme Clusters		○	○		●
RL2.3 Default Word Boundaries		●			
RL2.4 Default Case Conversion					
RL2.5 Name Properties	●	●	●		●
RL2.6 Wildcards in Property Values					
RL2.7 Full Properties	●		●		●
Level 3: Tailored Support	○		○		
RL3.1 Tailored Punctuation					
RL3.2 Tailored Grapheme Clusters					
RL3.6 Context Matching			○		
RL3.7 Incremental Matches	●				
RL3.9 Possible Match Sets	●				
RL3.11 Submatchers					

● = full support, ○ = partial support

Encoding Chains

- Sometimes data is transformed, not just (character) encoded:

Encoding Chains

- Sometimes data is transformed, not just (character) encoded:

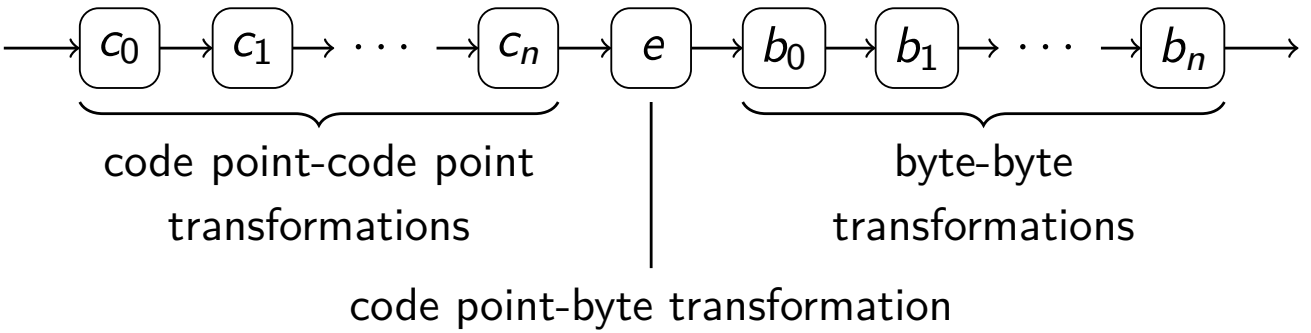
ROT-13	OCE	any substitution cipher
URL-encoding	XOR(n)	...

Encoding Chains

- Sometimes data is transformed, not just (character) encoded:

ROT-13	OCE	any substitution cipher
URL-encoding	XOR(n)	...

- We can generalize:

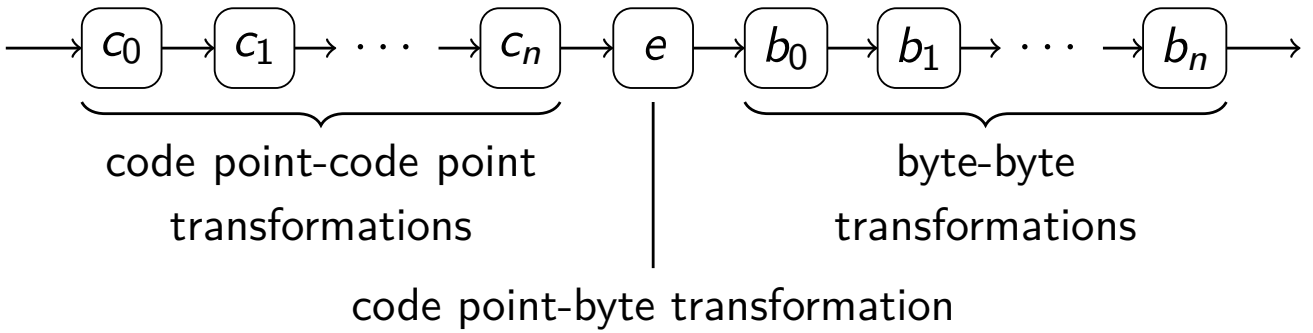


Encoding Chains

- Sometimes data is transformed, not just (character) encoded:

ROT-13	OCE	any substitution cipher
URL-encoding	XOR(n)	...

- We can generalize:



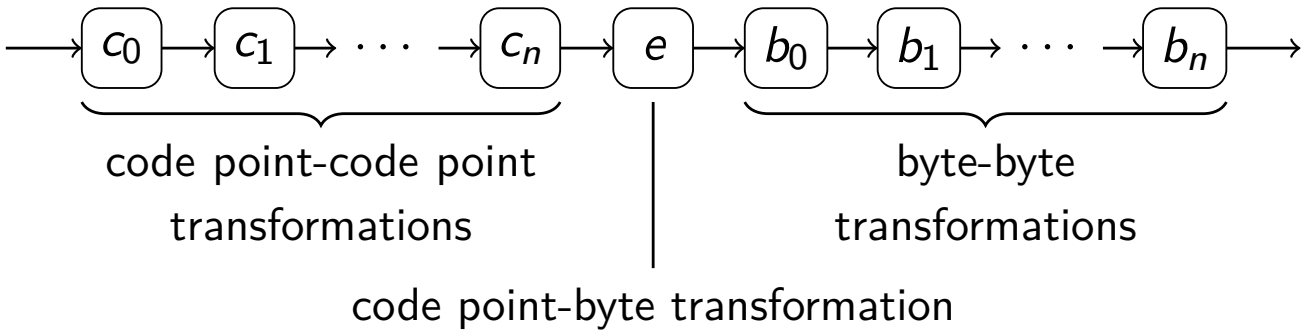
- We support arbitrarily-long encoding chains:

Encoding Chains

- Sometimes data is transformed, not just (character) encoded:

ROT-13	OCE	any substitution cipher
URL-encoding	XOR(n)	...

- We can generalize:



- We support arbitrarily-long encoding chains:

`UTF-16LE|OCE` `UTF-8|XOR(1F)`

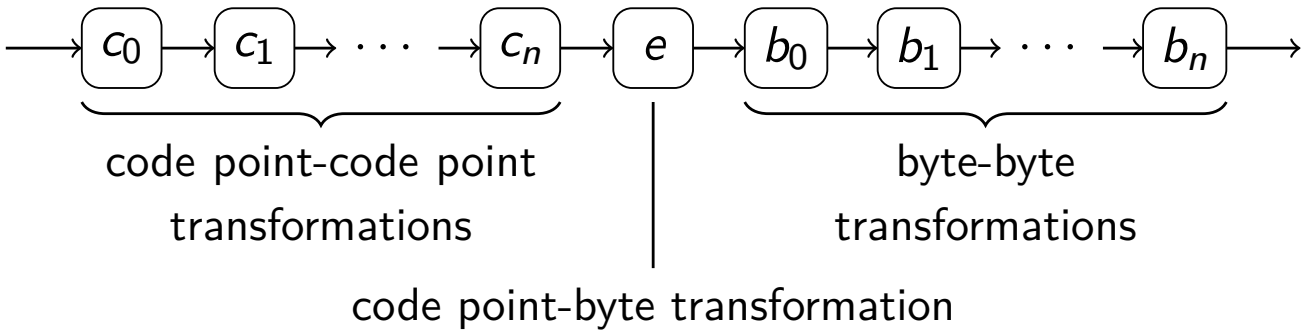
`ROT-13|ASCII|OCE|XOR(D7)`

Encoding Chains

- Sometimes data is transformed, not just (character) encoded:

ROT-13	OCE	any substitution cipher
URL-encoding	XOR(n)	...

- We can generalize:



- We support arbitrarily-long encoding chains:

`UTF-16LE|OCE` `UTF-8|XOR(1F)`
`ROT-13|ASCII|OCE|XOR(D7)`

- All the work happens at pattern-compile time.

Hit Context

- Seeing *decoded* context around search hits is handy

Hit Context

- Seeing *decoded* context around search hits is handy

A T T A C K A T D A W N

Hit Context

- Seeing *decoded* context around search hits is handy

A T T A C K A T D A W N

- But context could be *mojibake*!

Hit Context

- Seeing *decoded* context around search hits is handy

A T T A C K A T D A W N

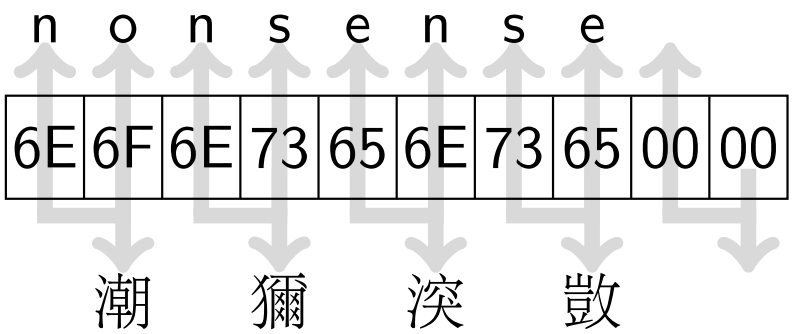
- But context could be *mojibake*!
- Is it nonsense or is it 潮獮滨敳?

Hit Context

- Seeing *decoded* context around search hits is handy

A T T A C K A T D A W N

- But context could be *mojibake*!
- Is it nonsense or is it 潮獮滾斀?

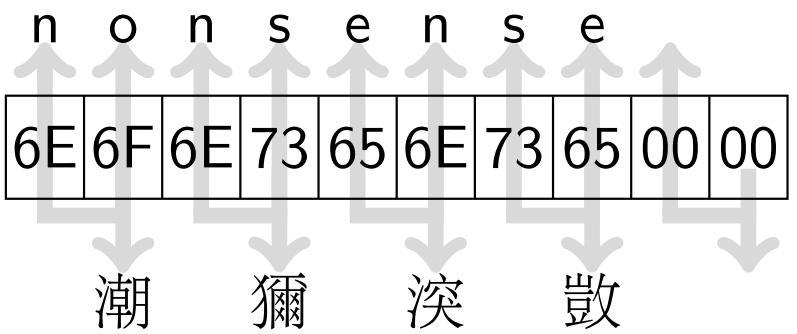


Hit Context

- Seeing *decoded* context around search hits is handy

A T T A C K A T D A W N

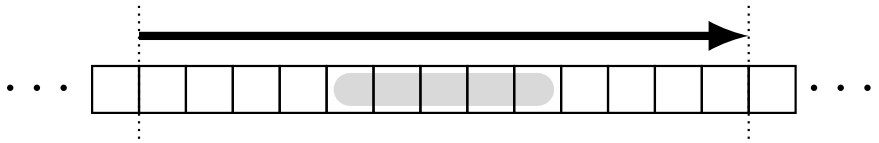
- But context could be *mojibake*!
- Is it nonsense or is it 潮獮滾斀?



- User discretion is advised.

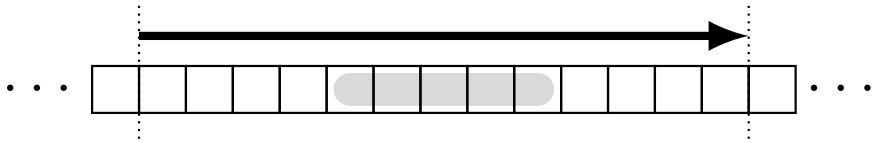
How We Decode Hit Context

Left to right



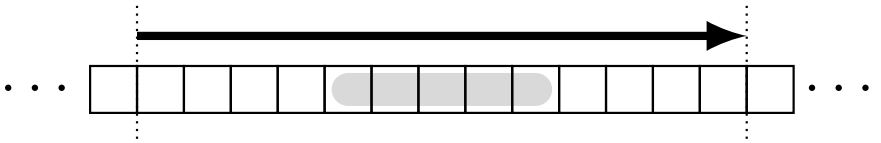
How We Decode Hit Context

Left to right



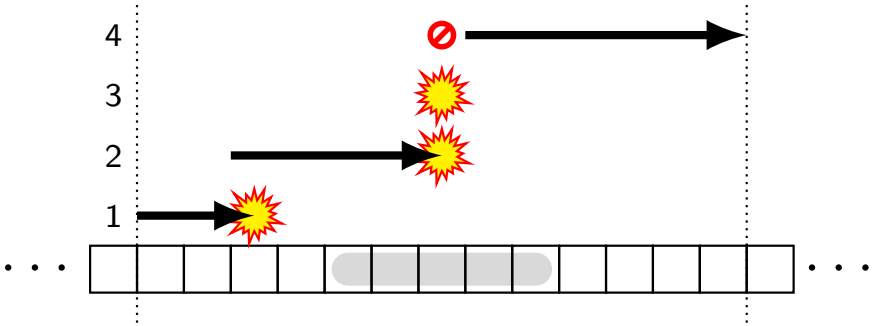
How We Decode Hit Context

Left to right



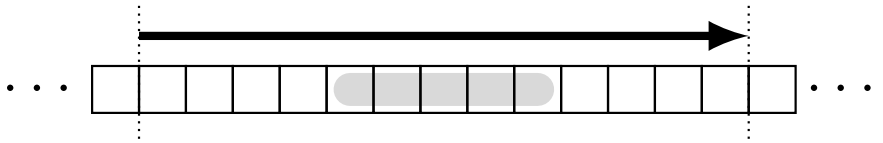
✘

Left to right, restarting on error



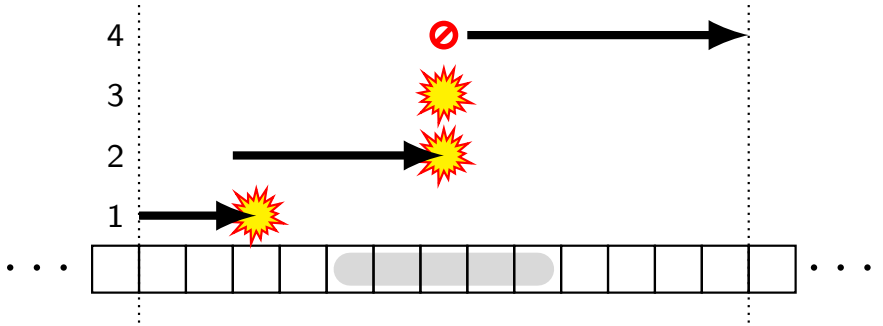
How We Decode Hit Context

Left to right



✘

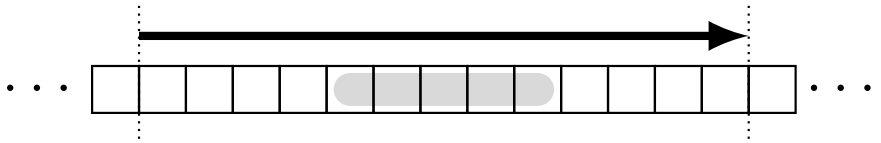
Left to right, restarting on error



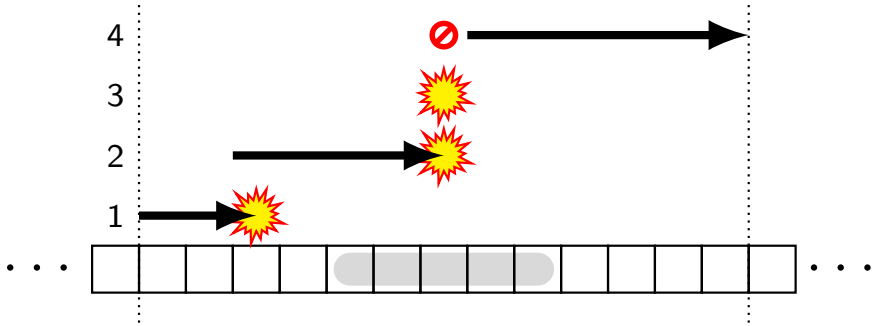
✘

How We Decode Hit Context

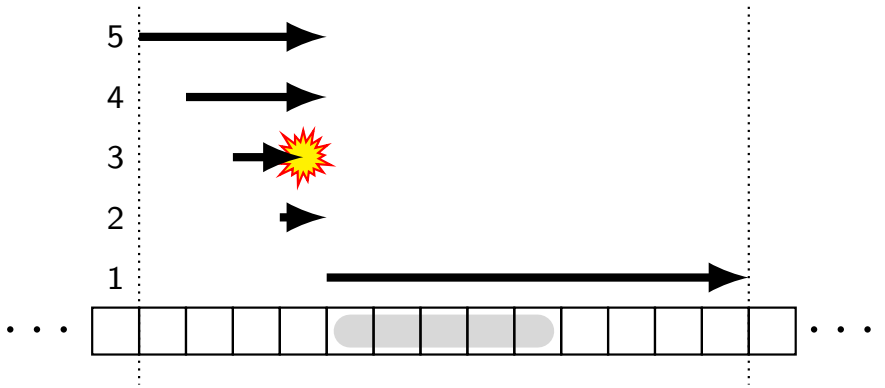
Left to right



Left to right, restarting on error

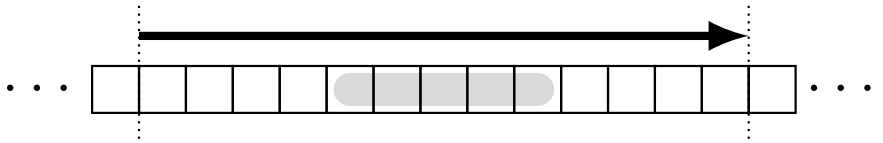


Leading context right to left, hit and trailing context left to right



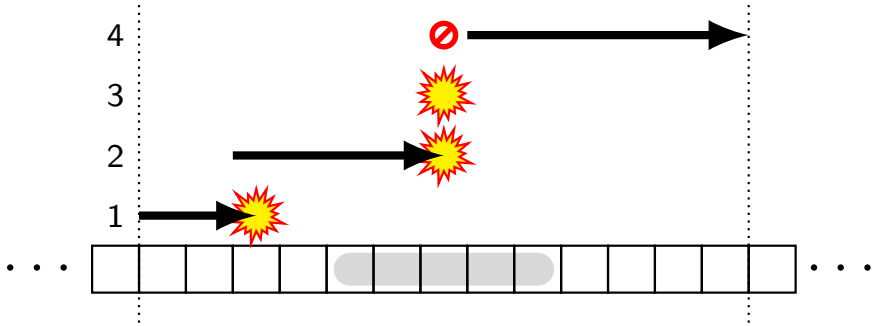
How We Decode Hit Context

Left to right



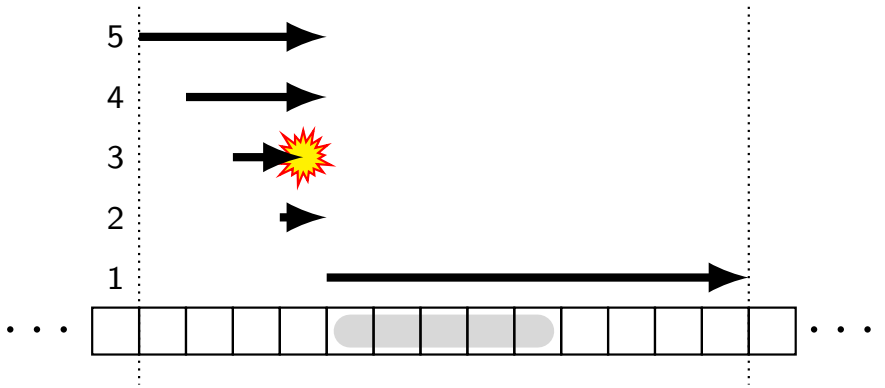
✘

Left to right, restarting on error



✘

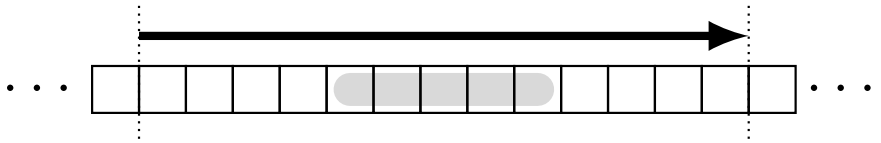
Leading context right to left, hit and trailing context left to right



✔

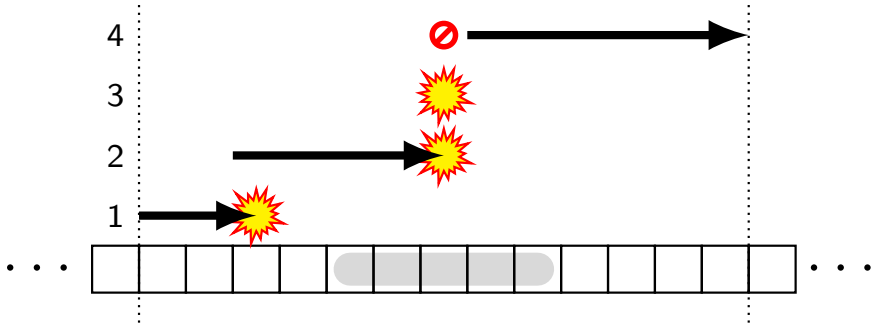
How We Decode Hit Context

Left to right



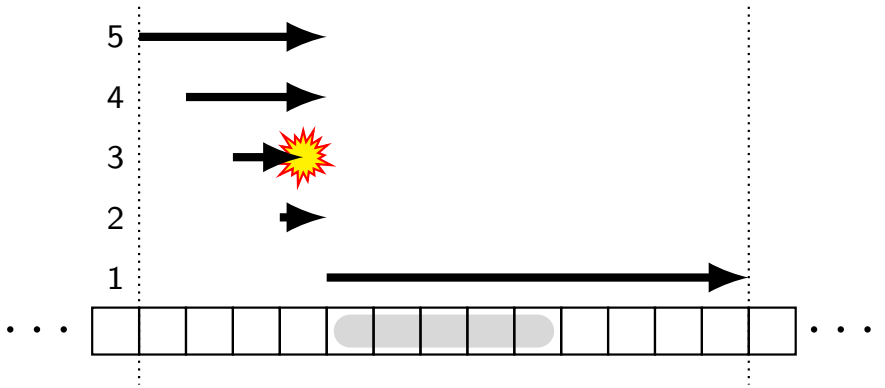
✘

Left to right, restarting on error



✘

Leading context right to left, hit and trailing context left to right



✔

$O(n^2)$ but can happen after (or in parallel with) search

How fast is search?

Image	Size	Hits	Bandwidth
Charlie-2009-11-12.E01	10GB	543,719	131.7 MB/s
Jo-2009-11-12.E01	14GB	536,906	123.8 MB/s
Pat-2009-11-12.E01	12GB	522,220	124.7 MB/s

Search performance using lightgrep via bulk_extractor
Images from the NPS 2009 M57 Patents-Redacted scenario

How fast is search?

Image	Size	Hits	Bandwidth
Charlie-2009-11-12.E01	10GB	543,719	131.7 MB/s
Jo-2009-11-12.E01	14GB	536,906	123.8 MB/s
Pat-2009-11-12.E01	12GB	522,220	124.7 MB/s

Search performance using lightgrep via bulk_extractor
Images from the NPS 2009 M57 Patents-Redacted scenario

Multiencoding/Unicode search can be fast.

Conclusions

- Unicode is complex because natural language is complex

Conclusions

- Unicode is complex because natural language is complex
- Encodings are messy for historical reasons

Conclusions

- Unicode is complex because natural language is complex
- Encodings are messy for historical reasons
- GORY DETAILS $\rightsquigarrow$ Don't implement this stuff yourself

Conclusions

- Unicode is complex because natural language is complex
- Encodings are messy for historical reasons
- GORY DETAILS $\rightsquigarrow$ Don't implement this stuff yourself

BUT:

Conclusions

- Unicode is complex because natural language is complex
- Encodings are messy for historical reasons
- GORY DETAILS $\rightsquigarrow$ Don't implement this stuff yourself

BUT:

- **You can *use***
  - **multiencoding support**
  - **multilanguage support**

without knowing much of anything about the details

Conclusions

- Unicode is complex because natural language is complex
- Encodings are messy for historical reasons
- GORY DETAILS $\rightsquigarrow$ Don't implement this stuff yourself

BUT:

- **You can *use***
  - **multiencoding support**
  - **multilanguage support**

  without knowing much of anything about the details

- Encoding chains make more data directly searchable

Conclusions

- Unicode is complex because natural language is complex
- Encodings are messy for historical reasons
- GORY DETAILS $\rightsquigarrow$ Don't implement this stuff yourself

BUT:

- **You can *use***
  - **multiencoding support**
  - **multilanguage support**

  without knowing much of anything about the details

- Encoding chains make more data directly searchable

- Hit context is handy, not entirely trivial to produce

Conclusions

A problem to think about

- Latin alphabet is used by dozens of languages, but with accent marks

e è é ê ë ē ě ė ę ě ê ë

A problem to think about

- Latin alphabet is used by dozens of languages, but with accent marks

e è é ê ë ē ě ė ę ě ê ë

- Unicode has *confusable* characters

A problem to think about

- Latin alphabet is used by dozens of languages, but with accent marks

e è é ê ë ē ě ė ę ě ê ë

- Unicode has *confusable* characters

A problem to think about

- Latin alphabet is used by dozens of languages, but with accent marks

e è é ê ë ē ĕ ė ę ě ê ë

- Unicode has *confusable* characters

A problem to think about

- Latin alphabet is used by dozens of languages, but with accent marks

e è é ê ë ē ě ė ę ǎ ê̌ ë̌

- Unicode has *confusable* characters

- Unicode has characters which *decompose* to character sequences

VIII ⇒ V·I·I·I IJ ⇒ I·J ﷺ ⇒ صلى الله عليه وسلم

A problem to think about

- Latin alphabet is used by dozens of languages, but with accent marks

e è é ê ë ē ě ė ę ǎ ê ë

- Unicode has *confusable* characters

- Unicode has characters which *decompose* to character sequences

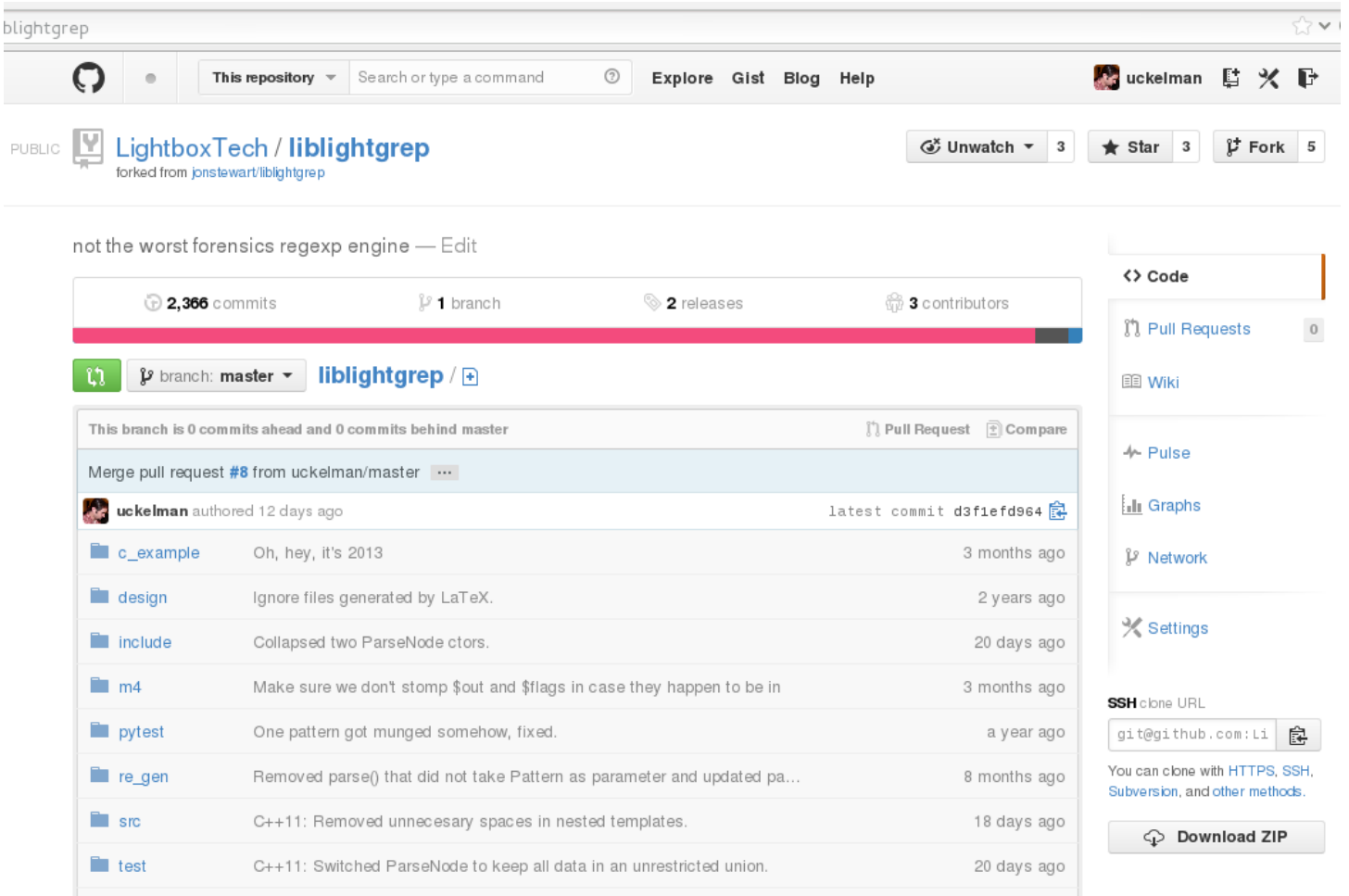
VIII ⇒ V·I·I·I IJ ⇒ I·J ﷺ ⇒ صلى الله عليه وسلم

- Wouldn't it be neat to have a "looks-like" operator or mode?

`\L{e}` `\L{VIII}` `\L{UNDUTCHABLES}`

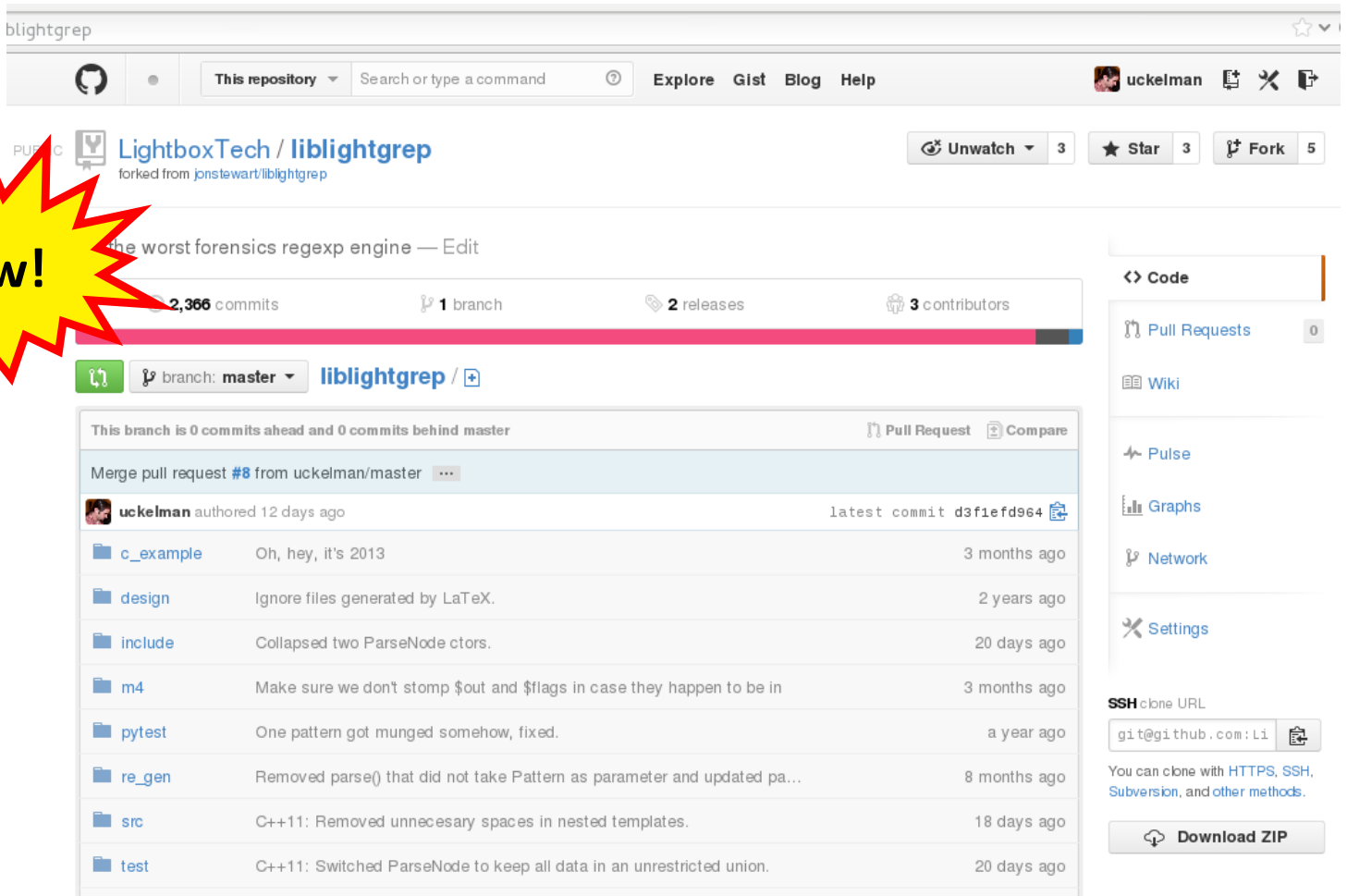
Where can I find this amazing software?

`https://github.com/LightboxTech/liblightgrep`



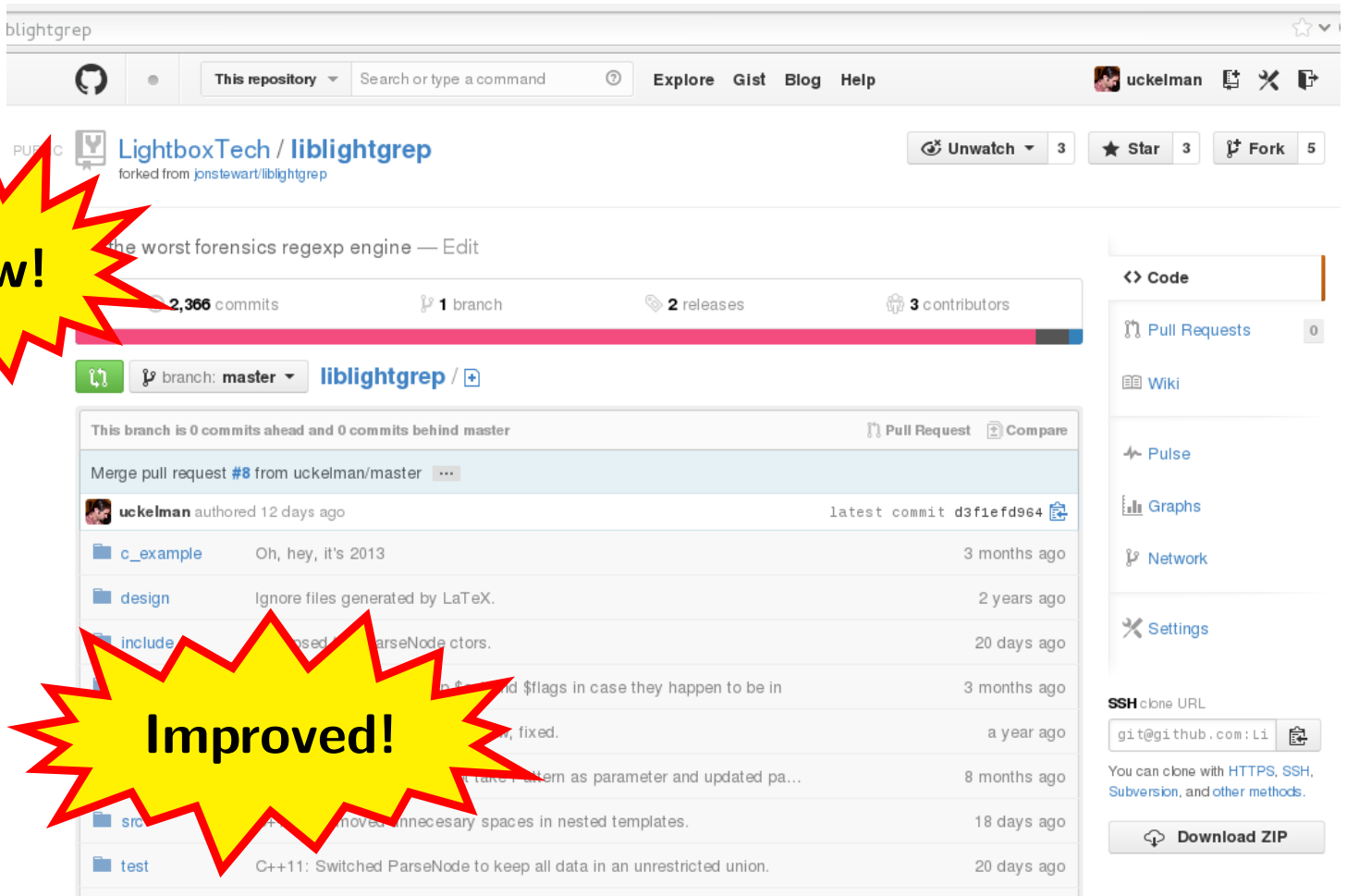
Where can I find this amazing software?

`https://github.com/LightboxTech/liblightgrep`



Where can I find this amazing software?

`https://github.com/LightboxTech/liblightgrep`

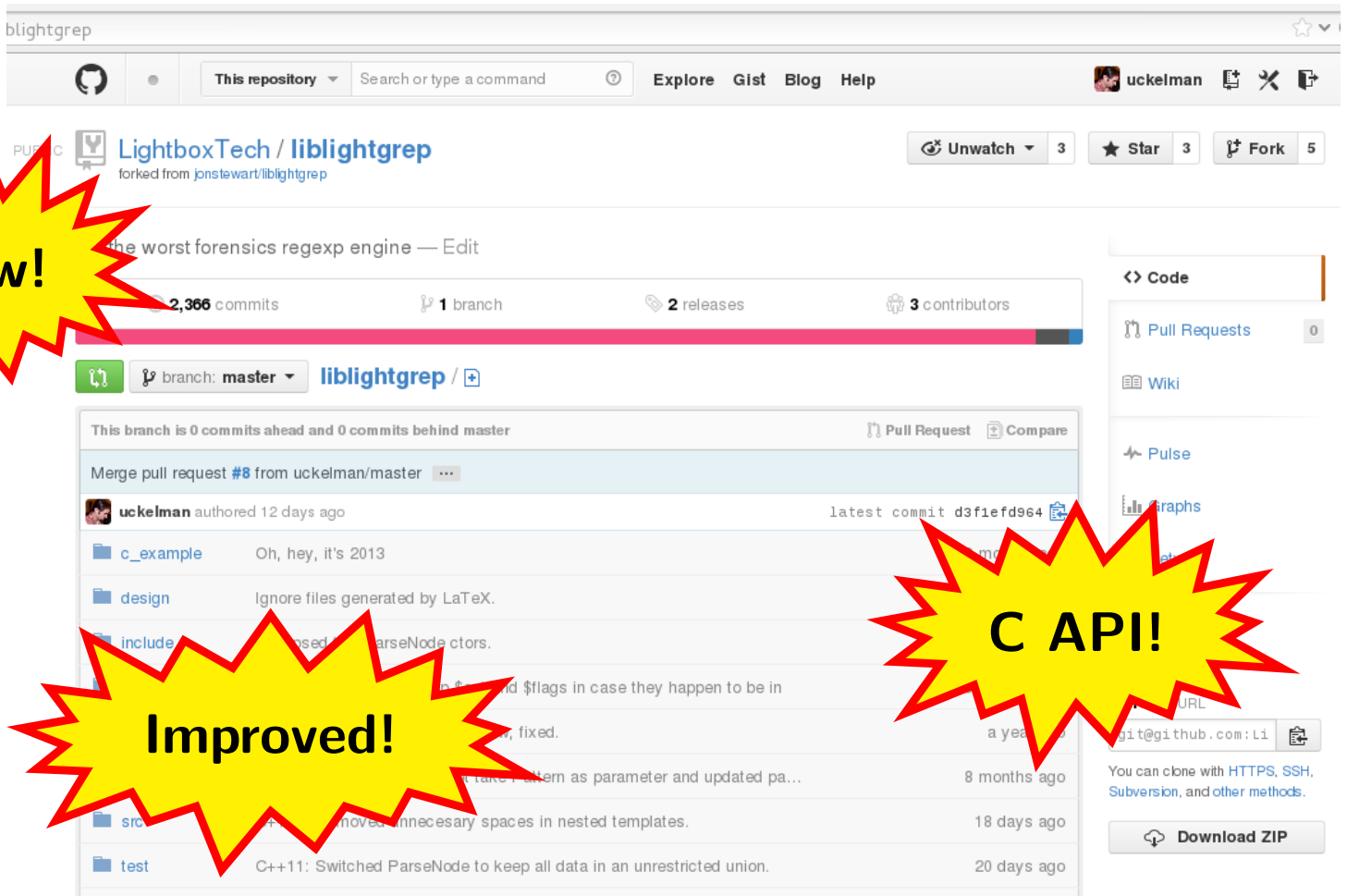


New!

Improved!

Where can I find this amazing software?

`https://github.com/LightboxTech/liblightgrep`



Where can I find this amazing software?

`https://github.com/LightboxTech/liblightgrep`