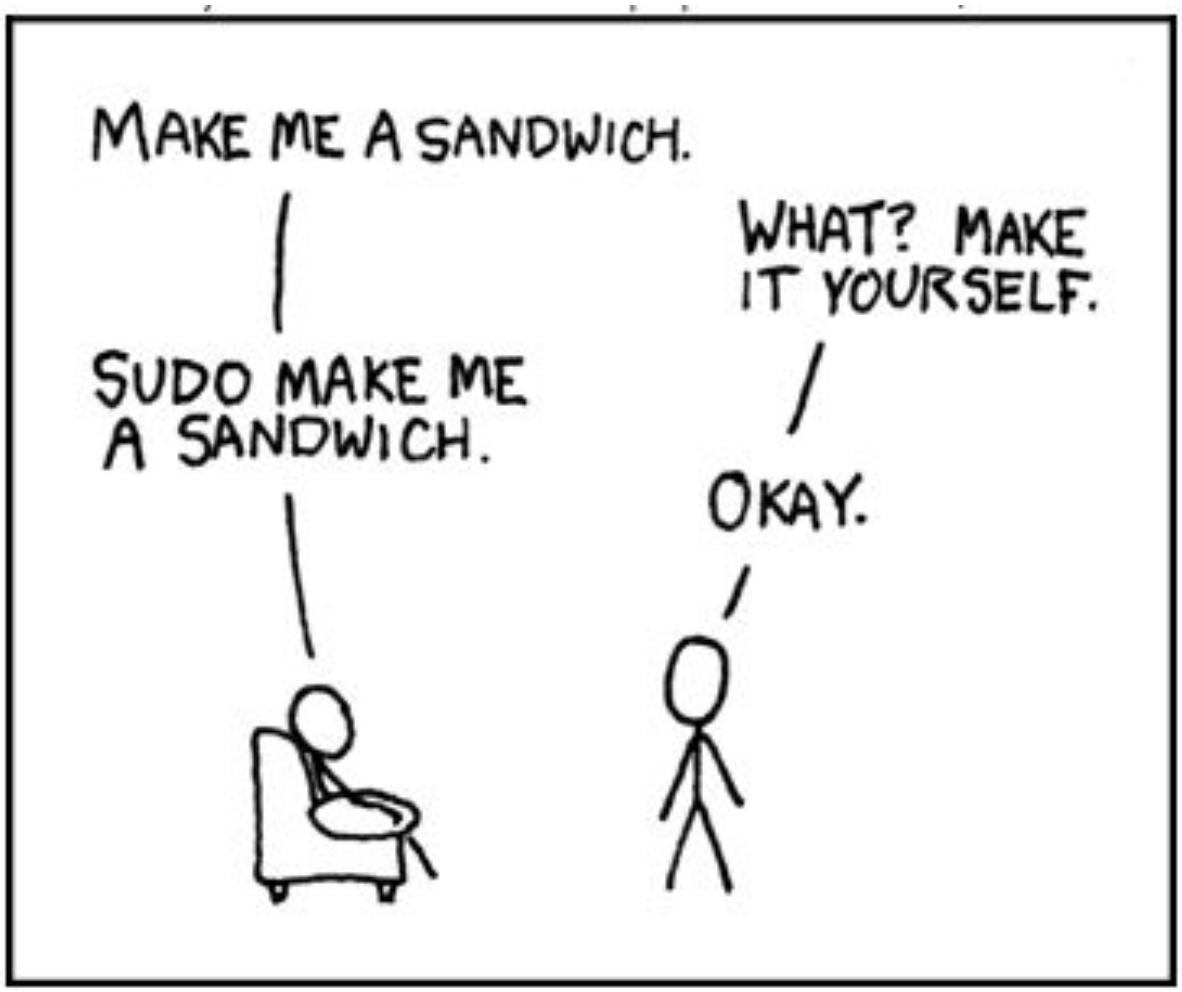

MAKE ME A SANDWICH.
SUDO MAKE ME A SANDWICH.
WHAT? MAKE IT YOURSELF.
OKAY.

RegEx

- RegEx: A **regular expression** is a special text string for describing a search pattern.
- Think of **regular expressions** as wildcards on steroids.
- Wildcard notations such as *.txt finds all text files in a directory.
- The regex equivalent is ^.*\.txt$.

Metacharacters

RE Metacharacter	Matches…
.	**Any one character, except new line**
[a-z]	**Any one of the enclosed characters (e.g. a-z)**
*	**Zero or more of preceding character**
? or \?	**Zero or one of the preceding characters**
+ or \+	**One or more of the preceding characters**

- any non-metacharacter matches itself

The grep Utility

- "grep" command:
  searches for text in file(s)

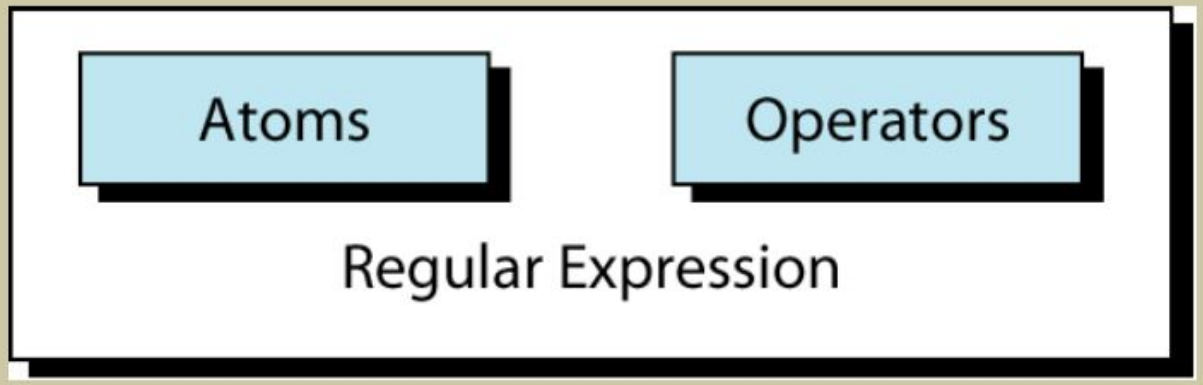
<u>Examples:</u>

```
% grep root mail.log
% grep r..t mail.log
% grep ro*t mail.log
% grep 'ro*t' mail.log
% grep 'r[a-z]*t' mail.log
```

more Metacharacters

RE Metacharacter	Matches…
^	**beginning of line**
$	**end of line**
char	**Escape the meaning of *char* following it**
[^]	**One character not in the set**
\<	**Beginning of word anchor**
\>	**End of word anchor**
( ) or \(\)	**Tags matched characters to be used later (max = 9)**
| or \\|	**Or grouping**
x\{m\}	**Repetition of character x, m times (x,m = integer)**
x\{m,\}	**Repetition of character x, at least m times**
x\{m,n\}	**Repetition of character x between m and m times**

Regular Expression

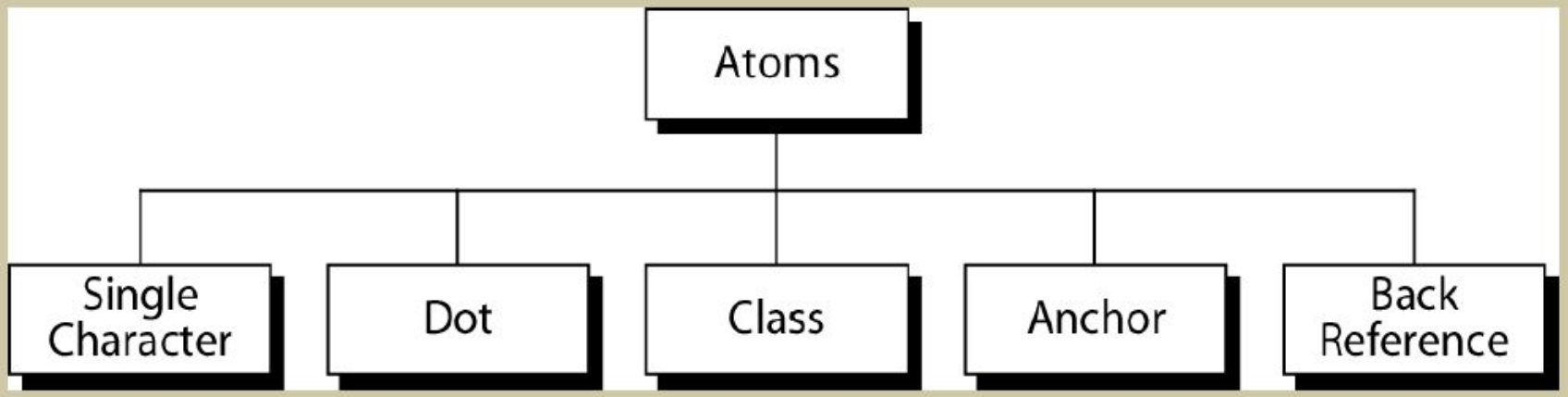


An atom specifies <u>what</u> text is to be matched and <u>where</u> it is to be found.

An operator combines regular expression atoms.

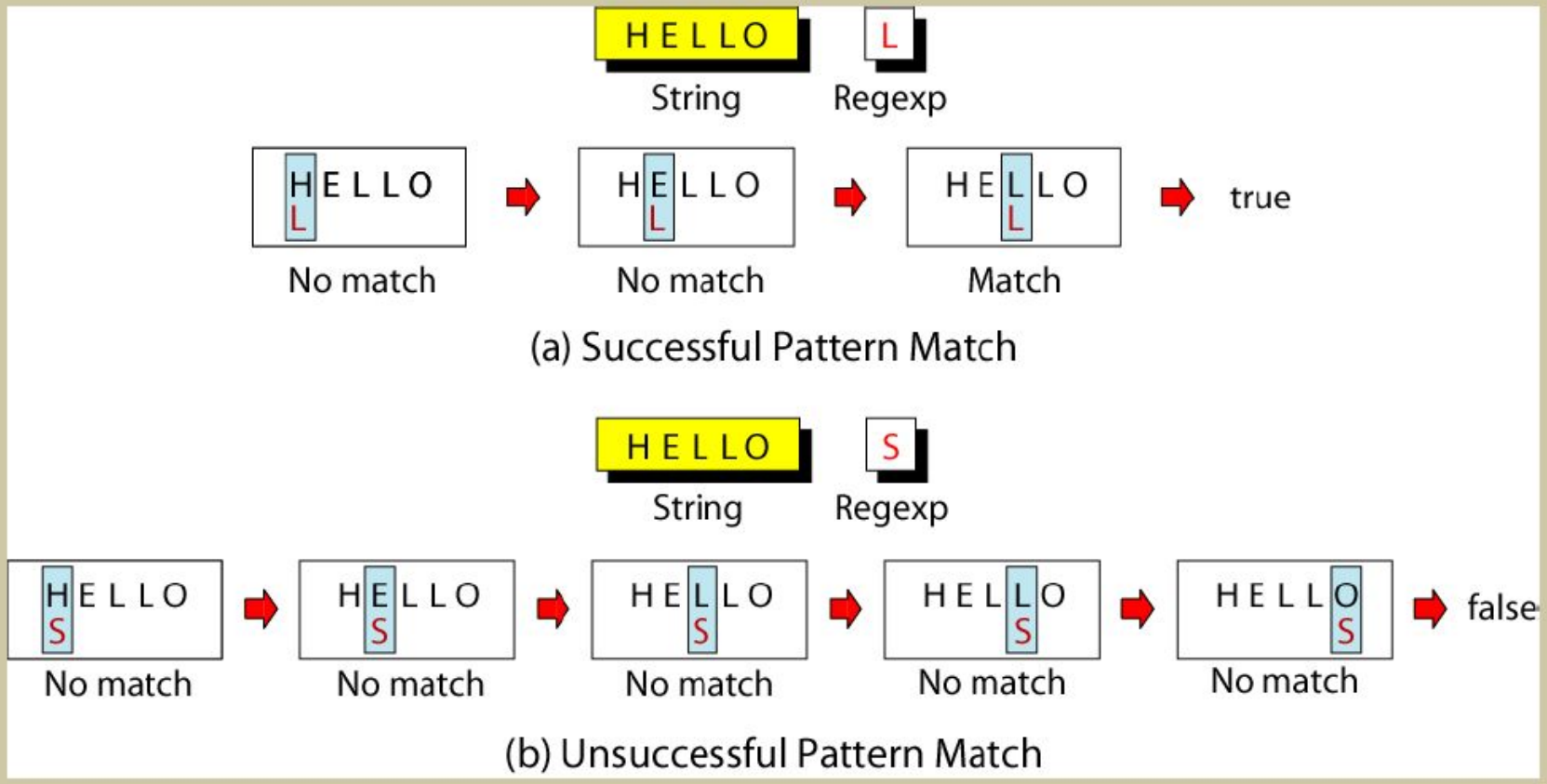
Atoms

An atom specifies <u>what</u> text is to be matched and <u>where</u> it is to be found.



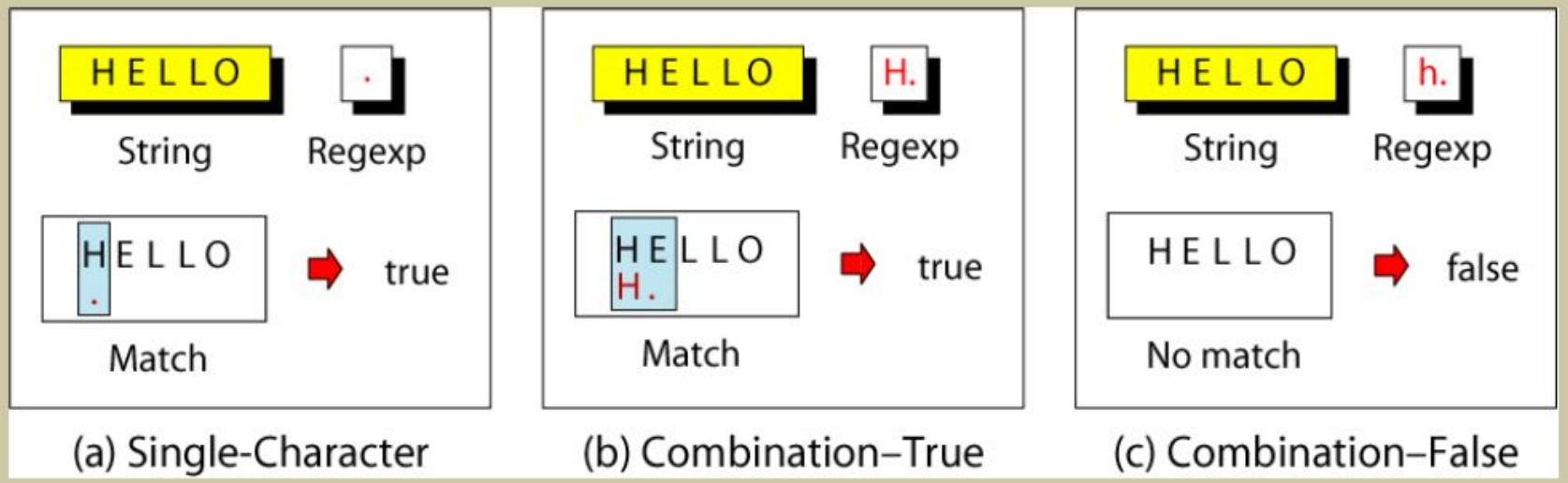
Single-Character Atom

A single character matches itself



Dot Atom

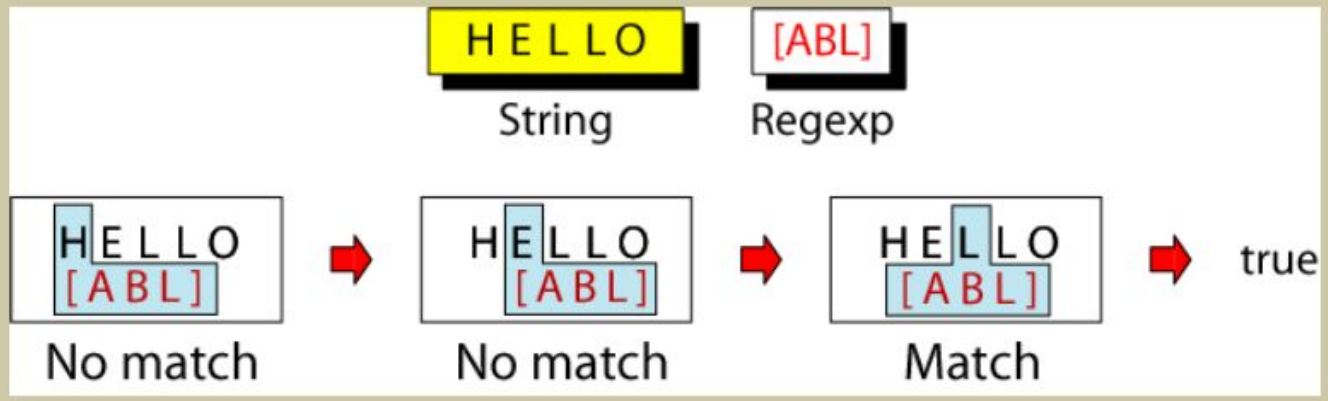
matches any single character except for a new line character (\n)



Class Atom

matches only single character that can be any of the characters defined in a set:

<u>Example:</u> [ABC] matches either A, B, or C.



Notes:

1) A range of characters is indicated by a dash, e.g. [A-Q]
2) Can specify characters to be excluded from the set, e.g. [^0-9] matches any character other than a number.

RegEx

RegExpr	Means	RegExpr	Means
[A-H]	[ABCDEFGH]	[^AB]	Any character except A or B
[A-Z]	Any uppercase alphabetic	[A-Za-z]	Any alphabetic
[0-9]	Any digit	[^0-9]	Any character except a digit
[[a]	[or a	[]a]	] or a
[0-9\-]	digit or hyphen	[^\^]	Anything except^

Anchors

Anchors tell **where** the next character in the pattern **must** be located in the text data.

Anchor	Means	Example
^	Beginning of line	One line of text.\n
$	End of line	One line of text.\n
\<	Beginning of word	One line of text.\n
\>	End of word	One line of text.\n

Sequence Operator

In a sequence operator, if a series of atoms are shown in a regular expression, there is no operator between them.

`dog`	→	matches the pattern "dog"
`a..b`	→	matches "a" , any two characters, and "b"
`[2-4][0-9]`	→	matches a number between 20 and 49
`[0-9][0-9]`	→	matches any two digits
`^$`	→	matches a blank line
`^.$`	→	matches a one-character line
`[0-9]-[0-9]`	→	matches two digits separated by a "-"

Alternation Operator: | or \|

operator (| or \|) is used to define one
or more alternatives

`UNIX|unix`	→	matches "UNIX" or "unix"
`Ms|Miss|Mrs`	→	matches "Ms" or "Miss" or "Mrs"

Note: depends on version of "grep"

Repetition Operator: \{...\}

The repetition operator specifies that the atom or expression immediately before the repetition may be repeated.

`\{m , n\}`

matches previous character m to n times.

`A\{3 , 5\}`	➡	matches "AAA" , "AAAA", or "AAAAA"
`BA\{3 , 5\}`	➡	matches "BAAA" , "BAAAA", or "BAAAAA"

Basic Repetition Forms

Formats

`\{m\}`	matches previous atom exactly m times
`\{m, \}`	matches previous atom m times or more
`\{, n\}`	matches previous atom n times or less

Examples

`CA\{5\}`	`CAAAAA`
`CA\{3,\}`	`CAAA, CAAAA, CAAAAA, …`
`CA\{,2\}`	`C, CA, CAA`

Short Form Repetition Operators: * + ?

Formats

`*`	special case: matches previous atom zero or more times
`+`	special case: matches previous atom one or more times
`?`	special case: matches previous atom 0 or one time only

Examples

`BA*`	B, BA, BAA, BAAA, BAAAA, . . .
`B.*`	B, BA . . . BZ, BAA . . . BZZ, BAAA . . . BZZZ, . . .
`.*`	zero or more characters
`.+`	one or more characters
`[0-9]?`	zero or one digit

Group Operator

In the group operator, when a group of characters is enclosed in parentheses, the next operator applies to the whole group, not only the previous characters.

Regexp		Matches
`A(BC)\{3\}`	➡	`ABCBCBC`
`(F(BC)\{2\}G)\{2\}`	➡	`FBCBCGFBCBCG`

Note: depends on version of "grep"
use \(and \) instead

RegEx

Further Reading:

- http://tldp.org/LDP/abs/html/x17129.html
- https://www.rexegg.com/regex-quickstart.html
- **http://misc.yarinareth.net/regex.html**

Awk

AWK

- a programming language designed for text processing
- Used for processing regular expressions in a script
- Used when the text is in file / delimited field format
- typically used as a data extraction and reporting tool
- a powerful standard feature of most Unix-like operating systems.

Awk

awk operations:

- scans a file line by line
- splits each input line into fields
- compares input line/fields to pattern
- performs action(s) on matched lines

Useful for:

- transforming data files
- Producing formatted reports

Programming constructs:

- format output lines for reports
- arithmetic and string operations
- conditionals and loops

Basic awk Script

- consists of patterns & actions:

```
pattern {action}
```

  - if pattern is missing, action is applied to all lines
  - if action is missing, the matched line is printed
  - must have either pattern or action

<u>Example:</u>

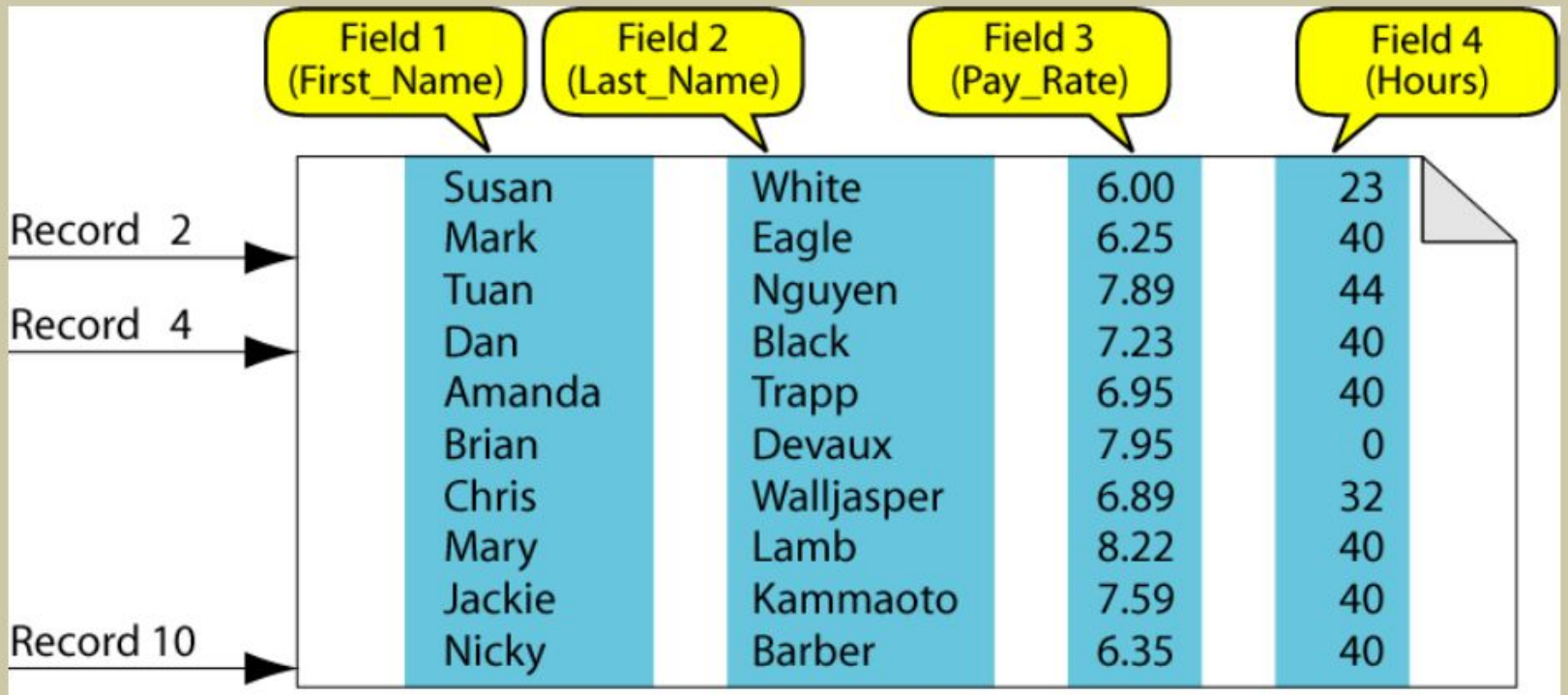
```
awk '/for/' testfile
```

  - prints all lines containing string “for” in testfile

Basic Terminology: input file

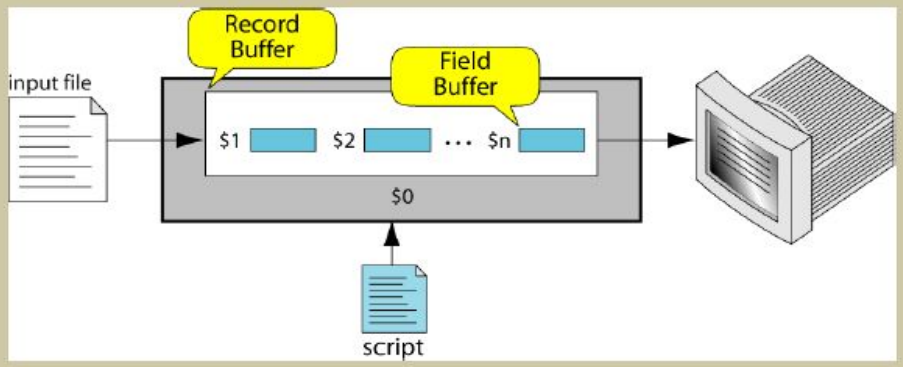
- A field is a unit of data in a line
- Each field is separated from the other fields by the field separator
  - default field separator is whitespace
- A record is the collection of fields in a line
- A data file is made up of records

Example Input File



A file with 10 records, each with four fields

Buffers



- awk supports two types of buffers: <u>record</u> and <u>field</u>
- field buffer:
  - one for each fields in the current record.
  - names: $1, $2, …
- record buffer :
  - $0 holds the entire record

Some System Variables

FS	Field separator (default=whitespace)
RS	Record separator (default=\n)
NF	Number of fields in current record
NR	Number of the current record
OFS	Output field separator (default=space)
ORS	Output record separator (default=\n)
FILENAME	Current filename

Example: Records and Fields

```
% cat emps
Tom Jones        4424    5/12/66 543354
Mary Adams       5346    11/4/63 28765
Sally Chang      1654    7/22/54 650000
Billy Black      1683    9/23/44 336500

% awk '{print NR, $0}' emps
1 Tom Jones      4424    5/12/66 543354
2 Mary Adams     5346    11/4/63 28765
3 Sally Chang    1654    7/22/54 650000
4 Billy Black    1683    9/23/44 336500
```

Example: Colon as Field Separator

```
% cat em2
Tom Jones:4424:5/12/66:543354
Mary Adams:5346:11/4/63:28765
Sally Chang:1654:7/22/54:650000
Billy Black:1683:9/23/44:336500

% awk -F: '/Jones/{print $1, $2}' em2
Tom Jones 4424
```

Pattern types

- match
  - entire input record
    regular expression enclosed by '/' s
  - explicit pattern-matching expressions
    ~ (match), !~ (not match)

- expression operators
  - arithmetic
  - relational
  - logical

Example: match input record

```
% cat employees2
Tom Jones:4424:5/12/66:543354
Mary Adams:5346:11/4/63:28765
Sally Chang:1654:7/22/54:650000
Billy Black:1683:9/23/44:336500

% awk -F: '/00$/' employees2
Sally Chang:1654:7/22/54:650000
Billy Black:1683:9/23/44:336500
```

Awk

Further Reading:

- www.hcs.harvard.edu/~dholland/computers/awk.html
- https://www.digitalocean.com/community/tutorials/how-to-use-the-awk-language-to-manipulate-text-in-linux

sed

sed

- a programming language designed for text processing
- Used for processing regular expressions in a script
- Used when the text is in a stream (no delimited field structure)
- typically used as a stream editor (thus the name)
- a powerful standard feature of most Unix-like operating systems

sed

sed operations:

- Loops through a file line by line
- Looks for text patterns
- Executes commands on a match
  - Substitute, delete, insert a new line, etc.

Useful for:

- Transforming data files
- Finding text strings and changing them

Programming constructs:

- A rather primitive language

sed

% cat example.txt

```
unix is a great os. unix is open source.
unix is easy to learn.
```

% sed ‘s/unix/linux/’ example.txt

```
linux is a great os. linux is open
source. linux is easy to learn.
```

% sed ‘s/unix/linux/2’ example.txt

```
unix is a great os. linux is open source.
unix is easy to learn.
```

sed

```
% cat example.txt
unix is a great os.
unix is open source.
unix is easy to learn.

% sed '3 s/unix/linux/' example.txt
unix is a great os.
unix is open source.
linux is easy to learn.
```

Further Reading:

- http://www.wikiwand.com/en/Sed
- https://www.tutorialspoint.com/sed/sed_overview.htm