LOGDNA EBOOK

Regular Expressions in Grep

If you've ever had to search, parse, or edit blocks of text programmatically, chances are you're familiar with regular expressions. This eBook will cover exactly what Regular Expressions are, what they're used for, benefits, and some examples.

logdna

WHAT ARE REGULAR EXPRESSIONS?

Regex is a special text string/language used for describing search patterns and matching strings in text. Its flexible and powerful syntax lets you create detailed search patterns, from simple words and phrases to complex constructs like e-mail addresses and phone numbers. It's much more powerful than a simple string comparison, and is almost universally supported across programming languages, frameworks, and text editors.

Linux comes with GNU grep command which supports regex. Grep stands for "global regular expression print". Grep is used to find what you're looking for, stored anywhere in the file system matching a specified pattern.

Simple Grep Examples

```
grep 'word' file1 file2 file3

grep 'username' /etc/passwd
```

You can use regex to specify a string of characters or pattern for grep to match instead of words.

BENEFITS OF REGEX

Benefits of Regex

Regexes are much more flexible than traditional text searches. They can detect almost any pattern of letters, numbers, symbols, special characters, and even metacharacters. Where traditional searches look for exact matches, regexes can match patterns of varying length. This makes them useful for finding constructs such as email addresses, IP addresses, URLs, and phone numbers.

Regexes are also concise. A single regex string can contain multiple search terms, perform multiple operations, and return multiple matches. This makes them very easy to implement, reuse, and modify.

Limitations of Regex

Regex has a steep learning curve. Even basic regular expressions are difficult to break down into their base operations. Compared to verbose languages like Python, understanding a regex requires a detailed understanding of the language. This can make expressions difficult to troubleshoot, especially for beginners. This is best expressed in the famous quote by Jamie Zawinski:

"Some people, when confronted with a problem, think 'I know, I'll use regular expressions.' Now they have two problems."

Heavy text processing can also be slow, depending on the complexity of the expression and the amount of text to search. There are ways to improve performance, such as using compiled expressions (the default in languages like Python), but it ultimately comes down to the efficiency of the expression.

HOW IS REGEX USED? USE CASES AND EXAMPLES:

Regex has a number of use cases, including:

Searching

Regex is designed for searching. Traditional search methods might only let you search for a specific string, but regex offers much more flexibility and control over how searches are performed.

Example:

Imagine you have a text document (such as a log file) and you want to find all instances of an email address appearing within the document. How would you go about this? You could start by searching for the "@" character, or for ".com", but what if the document also includes Twitter handles or website URLs? What about email addresses that end in ".edu", or ".net"? You would likely need to run multiple searches at a time and use complex string manipulation rules to extract out each potential match.

Alternatively, you could create a single regex expression that searches specifically for email addresses. One method is to use the following expression:

```
[a-zA-Z0-9-.!#$%&'*+\/=?^_`{|}~]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9]{2,3}
```

Let's break down this expression:

`[a-zA-Z0-9-.!#$%&'*+\/=?^_`{|}~]+`	Match any number of letters, numbers, or special characters listed.
`@`	Match the "@" symbol.
`[a-zA-Z0-9-]+`	Match any number of letters, numbers, or hyphen characters.
`\.`	Match a period.
`[a-zA-Z0-9]{2,3}`	Match any two or three letter word containing letters or numbers.

With this expression, we can return all instances of "user@example.com", "user.name@123company.co", or even "super_user+$10k@dash-co.net", but not "@example" or "http://example.com".

User Input Validation

Regex is often used as an input validation tool. Imagine you have a website where users can sign up by providing their email address. Before the registration can be completed, the user's email address must be verified. With regex, we can perform a simple validation test that checks the formatting of the user's address before we allow them to register. We can even use JavaScript to perform this test and notify the user in real time, while using the same expression used in the previous example.

String Replacement and Masking

We discussed how regex can be used to find patterns of text within larger documents. But what if you wanted to replace, mask, or delete certain text?

Example:

Consider a payment processing service that occasionally logs sensitive data such as credit card numbers and bank account details. To protect their users' privacy, the service should automatically scrub this data before sending its logs to a centralization service. But how do we detect and erase this data after the log has already been written?

> With regex, we can create expressions to detect numbers matching the formats used by credit card vendors. We can then use a method like Python's `re.sub()` to substitute each instance with another value.

Using Regex in LogDNA's Stream Editor

Log messages don't always appear perfectly formatted. This is why the LogDNA web app includes a stream editor feature that lets you change the formatting of your log data in real-time. You can use a regular expression as your search term, as well as toggle case sensitivity and global searching. This works similar to the sed command, while also formatting live log data.

Example:

Imagine you have an application that writes multiline

logs to syslog. To avoid generating multiple syslog events from a single application event, the syslog service automatically escapes newline characters. This ensures each event only writes a single syslog message, but this makes the log stream appear cluttered and difficult to read. With LogDNA, we can use the search and replace feature to find and replace all instances of the escaped newline character with an actual newline character:

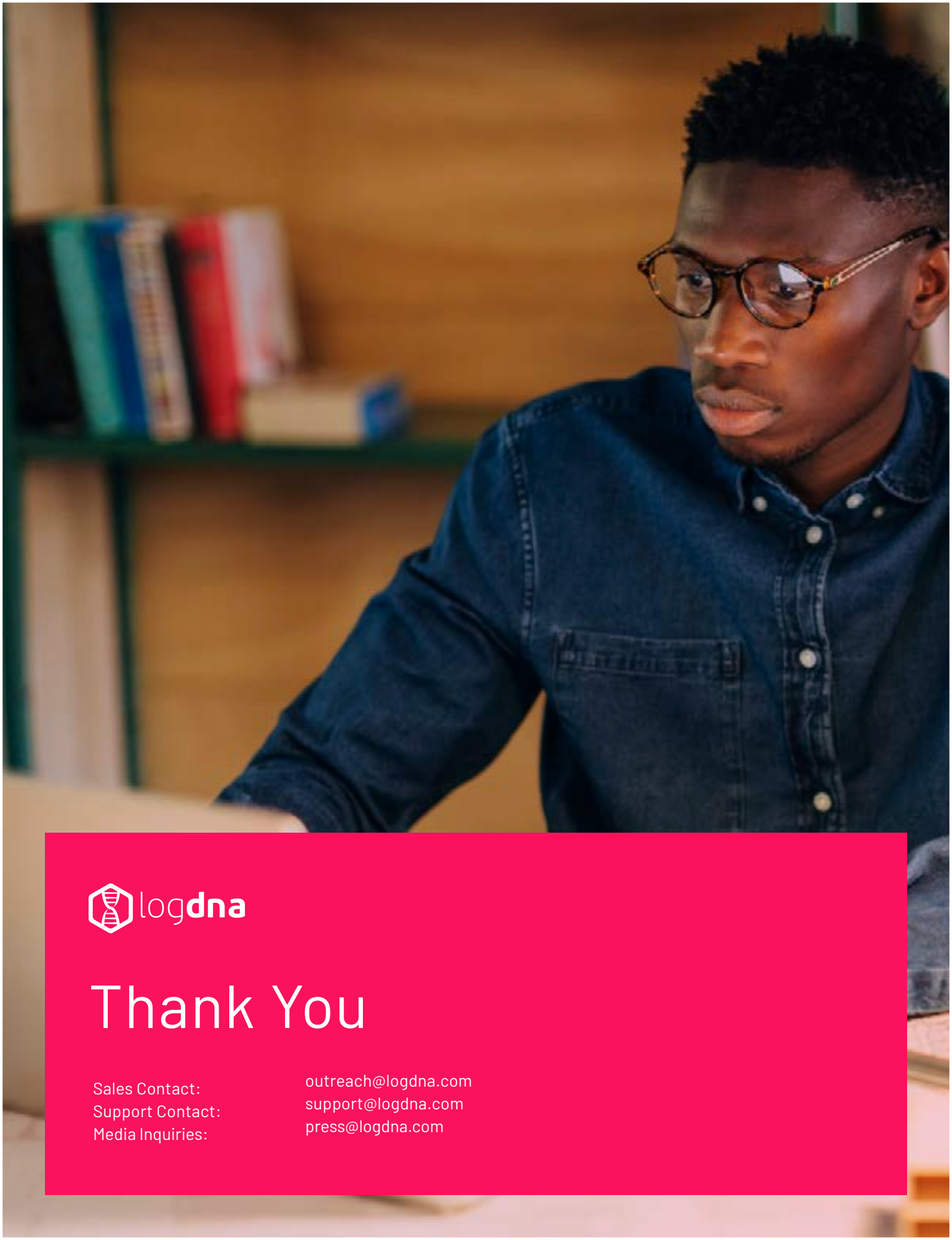
The "i" button toggles case sensitivity for the regular expression, while the "g" button toggles global or local matching. If global matching is disabled, only the first match in the stream is replaced. Clicking on the check mark performs the replace, and clicking on the "x" reverts it. Now, any current and new syslog messages will be displayed over multiple lines while leaving the actual log data untouched.

Conclusion

Despite being almost thirty years old, regex is still unfamiliar and esoteric territory for many developers. However, its flexibility and ubiquity make it a valuable addition to any developer's toolkit. If you want to learn more about regex or practice creating different expressions, sites like RegExr and regex101 provide interactive editors. Regular-Expressions.info also provides detailed tutorials, examples, and quick start guides.

About LogDNA

LogDNA is a centralized log management solution that helps modern engineering teams be more productive in a DevOps-oriented world. It enables frictionless consumption and actionability of log data so developers can monitor, debug, and troubleshoot their systems with ease.



logdna

Thank You

Sales Contact: outreach@logdna.com
Support Contact: support@logdna.com
Media Inquiries: press@logdna.com