CSE 20289 – Lecture 9

Pipelining, Regular Expressions

Monday, September 11th, 2023

Agenda – Lecture 9

- **Assignment 3**
  - Protect the Castle
  - Malicious URLs / Sensitive Info
  - Archive Extraction
- **Regular Expressions**
  - What are they?
  - Coding, coding, coding

When	Info
This Week – Week 4	Quiz – Week 4 M – Pipelines, Regular Expressions W – Regular Expressions F – RegEx / Filtering
Friday / Saturday	Assignment 3 4 to 5 hours
Friday / Saturday	Readings – Week 4
Next Sunday / Monday	Quiz – Week 4
Next Week	

Assignment 3 – Protect the Castle

USD 4.45 million

The global average cost of a data breach in 2023 was USD 4.45 million, a 15% increase over 3 years.

51%

51% of organizations are planning to increase security investments as a result of a breach, including incident response (IR) planning and testing, employee training, and threat detection and response tools.

USD 1.76 million

The average savings for organizations that use security AI and automation extensively is USD 1.76 million compared to organizations that don't.

https://www.ibm.com/reports/data-breach

Assignment 3 – Archives

- Check e-mail attachments (archives) for bad things
  - Malicious URLs
  - Sensitive Information – SSN / Sensitive Marking
- Big Loop
  - Monitor a directory for new archives (toscan)
  - Extract the archive
  - Check the archive contents for
    - Malicious URLs from a bad URL list
    - Sensitive information – SSN or *SENSITIVE*
  - Put the archive in either
    - Approved directory
    - Quarantined directory (with a reason)

Break it into pieces

Create a tinker directory and write small scripts

Write the big script to bring it altogether

Extra credit – nested archives

Estimated Time: 3 to 6 hours

Prompt - Discussion

How could you extract the date of a commit for a particular assignment?

Motivating Questions – This Week

1. Why is the **pipeline** pattern so powerful?
2. How do we use **regular expressions** to match text?
3. How do we **translate** characters?
4. How do we **extract** fields?
5. How do we **search** for patterns?
6. How do we **modify** text streams?
7. How do we perform more complex **text processing**?

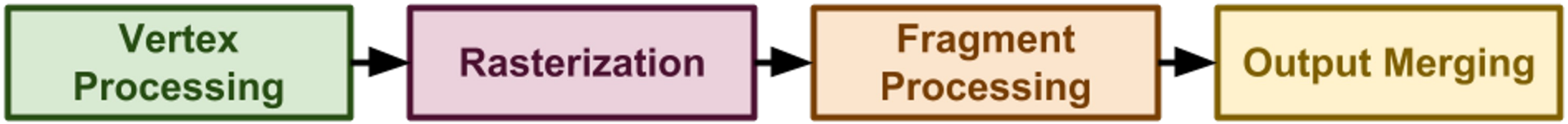
Pipeline: Assembly Line

A **Unix pipeline** is a **computational assembly line** where the output of one **process** is fed to the next **process**:

- Each process is tasked with a **single operation**
- Each operation performs an **action** on the previous input
- Each operation is **independent** and can be **chained** in many different ways

Pipeline: Powerful Pattern

The **pipeline pattern** is found throughout **Computer Science**:



Pipeline: Demonstration

On **student10.cse.nd.edu**:

1. How many instances of **bash**? **csh**? **sh**?

2. How many different types of **shells** are being used?

3. Who has the most **processes**?

L33t is not always neat

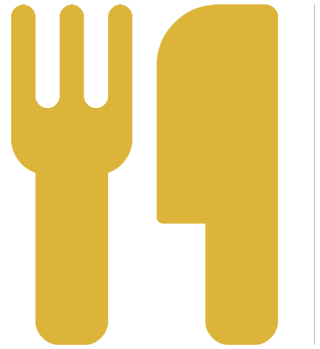
RegEx / Pipelining is Neat – Must be Understandable

Slice and Dice

1. What are **regular expressions**?

2. What are **filters?**

3. How do **regular expressions** and **filters** combine to manifest the **Unix Philosophy**?



Regular Expression: Overview

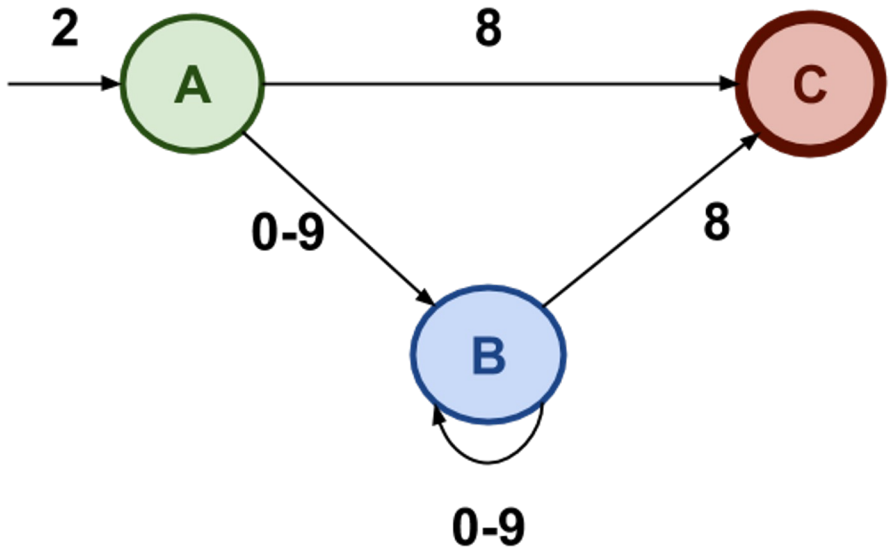
A **regular expression** (aka **regex**) is a sequence of characters that define a search pattern that is used to match strings.

```
grep -i ' user:' /etc/passwd
grep -E ' (user|User):' /etc/passwd
grep -E ' [uU]ser:' /etc/passwd
```

- Based on formal language theory
- Many different implementations and syntaxes (vary between tools)

Regular Expression: Theory

A **regular expression** is a specification of a regular language that can represented by a **finite automaton (FA)**




```
grep -E '2[0-9]*8' /etc/passwd
```

Extended grep capabilities

Regular Expression: POSIX

Unix utilities such as **grep**, **sed**, and **awk** support the **POSIX Basic Regular Syntax (BRE)**:

- **Metacharacters** such as `()|{}[]` need to be escaped:

```
grep ' \(user\|User\):' /etc/passwd
```

- The **Extended Regular Syntax (ERE)** doesn't require escaping and supports additional features

```
grep -E ' (user|User):' /etc/passwd
```

Metacharacter	Description
.	Match any single character
*	Match preceding element zero or more times
?	Match preceding element zero or one time
+	Match preceding element one or more times
[]	Matches a single character contained within bracket
[^]	Matches a single character not contained within bracket
^	Matches the starting position within string
$	Matches the ending position of the string
()	Marks subexpression that can be recalled later
|	Match either expression
\n	Matches the nth marked subexpression matched
{m, n}	Matches the preceding element between m to n times

Class	Character Set	Description
`[:alnum:]`	`[A-Za-z0-9]`	Alphanumeric
`[:alpha:]`	`[A-za-z]`	Alphabetic
`[:blank:]`	`[ \t]`	Space and tab
`[:space:]`	`[ \t\r\n\v\f]`	Whitespace
`[:digit:]`	`[0-9]`	Digits
`[:punct:]`	`[][!"#$%&'()*+,./:;<=>?@\^_`{|}~-]`	Punctuation
`[:lower:]`	`[a-z]`	Lowercase letters
`[:upper:]`	`[A-Z]`	Uppercase letters

Regular Expression: Examples

Given:

```
pikachu
bulbasaur
charmander
chespin
squirtle
meowth
togepi
oshawott
abra
jigglypuff
```

Write a regex to match:

1. All the strings
2. Only charmander and chespin
3. All the words with two t's
4. Words that don't start with a vowel
5. All words with two consecutive vowels
6. All words with two consecutive letters (same)
7. All words that begin and end with the same letter
8. All words with exactly 2 of r, s, or t

Filters

Filter: Overview

Unix includes many utilities that:

1. **Read** data from standard input
2. **Perform** some operation
3. **Write** results to standard output:

We call these utilities **filters** since they transform their inputs in some manner.

Filter: Tr

`tr` allows us to translate from one set of a characters to another set.

```
$ echo 'burn this city' | tr 'abc' 'xyz'
yurn this zity

$ echo 'burn this city' | tr 'a-z' 'A-Z'
BURN THIS CITY

$ echo 'burn this city' | tr -d '[:space:]'
burnthiscity
```

Filter: Cut

`cut` allows us to extract portions from each line of text:

```
$ echo 'burn this city' | cut -d ' ' -f 1
burn

$ echo 'burn this city' | cut -d ' ' -f 1,3
burn city

$ echo 'burn this city' | cut -c 2-4
urn
```

Activity: Dos2Unix

When transferring files from Windows to Unix, sometimes you have problems with **DOS line endings** (ie. **\r\n** instead of **\n**):

```
echo "hello, world"^M
```

Use **tr** to remove the **'\r'** character

```
tr -d '\r' < dos.txt > unix.txt
```

Filter: Grep

grep allows us to search for patterns:

```
$ grep '\(2[0-9]*8\)' /etc/passwd      # BRE

$ grep -E '(2[0-9]*8)' /etc/passwd     # ERE

$ grep -P '(2\d*8)' /etc/passwd        # Perl

$ grep -o '2[0-9]*8' /etc/passwd       # Show match

$ grep -v '2[0-9]*8' /etc/passwd       # Inverse
```

Activity: Contact Harvesting

Perform the following, given the webpage:

```
curl -sL https://cse.nd.edu/about-cse/administration-and-staff/
```

1. Extract all the **phone numbers**

   574-631-7388

2. Extract all the **email addresses**

   rbualuan@nd.edu

3. Extract all the **"Assistant" positions**

   Administrative Assistant

Filter: Sed

`sed` allows us to modify streams of text:

```
# Replace pikachu with raichu
$ echo "ash loves pikachu" | sed 's/pikachu/raichu/'

# Replace /var/lib with /tmp
$ cat /etc/passwd | sed -E 's|/var/lib|/tmp|g' | grep tmp

# Replace /var/… with /tmp/…
$ cat /etc/passwd | sed -E 's|/var/([^:]+)|/tmp/\1|g' | grep tmp

# Extract all fields with word user
$ cat /etc/passwd | sed -En 's|.*:(.*[uU]ser[^:]*):.*|\1|p'
```

Filter: Sed (More)

```
# Remove leading whitespace
echo "     Gotta collect them all" | sed -E 's/^\s+//'

# Remove leading and trailing whitespace
echo " Gotta collect them all " | sed 's/^[ \t]*//;s/[ \t]*$//'

# Remove lines with nologin
cat /etc/passwd | sed '/nologin/d'

# Print first 10 lines
cat /etc/passwd | sed 10q
```

Solutions

Activity: CS Curriculum

Answer the following questions, given the webpage:

```
curl -s https://cse.nd.edu/undergraduate/computer-science-curriculum/
```

1. How many **MATH** vs **PHYS** vs **CSE** courses?
2. How many **credits per semester**?
3. How many **sophomore CSE** courses?
4. How many **sophomore CSE** credits?
5. How many **different types** of **electives**?

Activity: `fix_indents.sh`

A common pet peeve is whether or not to use **tabs** or **spaces** to indent code and how wide those indents should be. For instance, some folks use tabs that are 8-spaces wide and others prefer indents to be just two spaces.

Because manually re-indenting people's code is a tedious and cumbersome task, write a script that lets you **replace tabs with spaces or vice versa**:

```
$ ./fix_indents.sh -t spaces -w 4 < source.sh
```

The command above will use the script to convert tabs into spaces that are four spaces wide.

Filter: Awk

`awk` is a powerful pattern matching language.

Each line is broken up into **fields**:

```
$0    [Hello,     | World]!
       $1           $2
```

Each script consists of the following **blocks**:

```
BEGIN       {}          # Executes at start
LINE        {}          # Executes for each line
END         {}          # Executes at end
```

Filter: Awk (Examples)

```
# List all PIDs
$ ps ux | grep $(whoami) | awk '{print $2}'


# Who has the most processes?
$ ps aux | awk '{print $1}' | awk '
                { names[$1] += 1}
        END     {
            for (name in names) {
                                print names[name], name
                }
      }' | sort -rn
```

Filter: Awk (Examples)

```
# Compute sum of numbers from 1 - 9
$ seq 1 9 | awk '
    BEGIN       { sum = 0   }
            { sum += $0 }
    END         { print sum }'

# Compute the average of numbers from 1 - 9
$ seq 1 9 | awk '
    BEGIN   { sum  = 0
              line = 0
            }
            { sum += $0
              line += 1
            }
        END     { print sum/line }
'
```

Activity: Advent of Code 2020 (Day 1)

Use shell scripting and UNIX filters to solve Day 1 of the Advent of Code (2020)!

Solution Part 1

Solution Part 2