SALIM HABIB UNIVERSITY

Operating System

SimpleFetch: A Bash-based HTTP Request Tool

Group Members:

M. Rabyaan
Khizar Bin Nasir
Osama Farooqi
Sahil SIndhoo

Table of Contents

Title: SimpleFetch: A Bash-based HTTP Request Tool

Introduction

SimpleFetch is a lightweight HTTP/HTTPS request tool built using Bash scripting. It focuses on providing essential functionalities such as GET/POST requests, caching, log management, and system monitoring. Designed to showcase Operating Systems concepts like threading, process management, and file handling, it is a practical and efficient tool for command-line users.

Problem Statement

The primary goal of this project is to provide an easy-to-understand alternative to existing HTTP request tools while simultaneously serving as a learning tool for Operating Systems concepts. The challenge was to create a tool that:

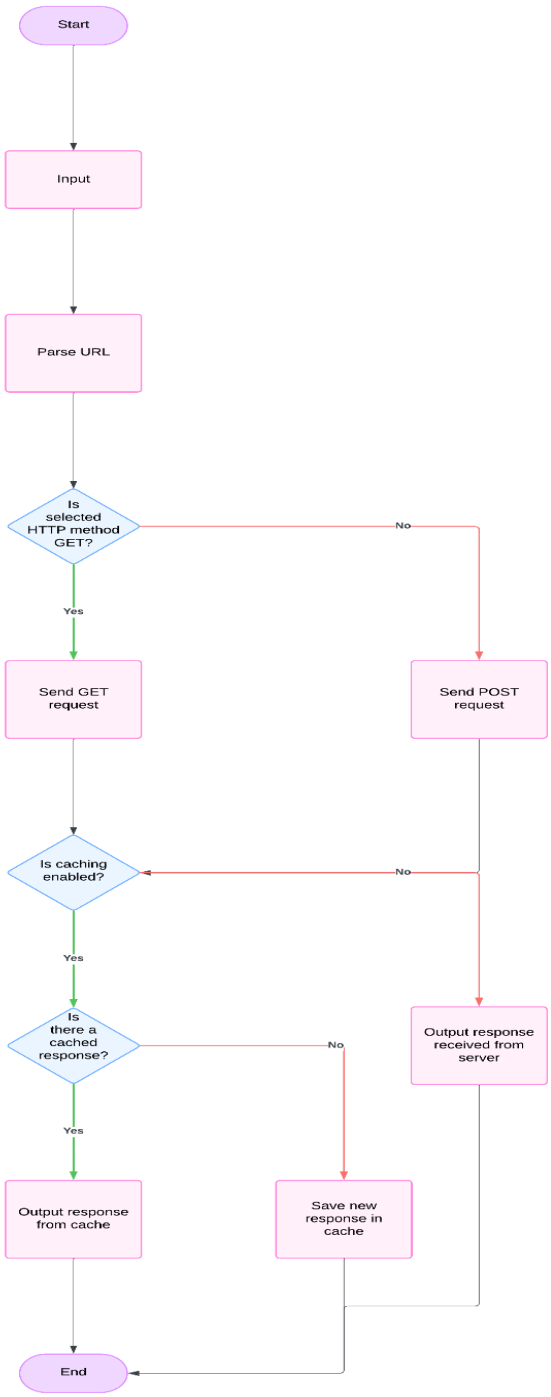
1. Simplifies HTTP/HTTPS requests for beginners.
2. Provides essential features like concurrent requests, caching, and log management.
3. Demonstrates OS-related concepts in a practical, real-world scenario.
4. Remains lightweight and straightforward compared to advanced tools like `curl` and `wget`.

Methodology

a) Stepwise Explanation

1. **Parsing Inputs**
    - The script accepts arguments such as the HTTP method (GET/POST), URL, optional data (for POST), and flags (e.g., `-c` for caching).
    - Example: `./simplefetch.sh GET http://example.com -c`
2. **URL Parsing**
    - Extracts components like protocol, host, path, and port using `awk` and `grep`.
3. **Handling HTTP Requests**
    - For GET requests:
        - Constructs the HTTP GET request and sends it using `nc` or `openssl`.
    - For POST requests:
        - Adds a Content-Length header and includes the data in the request body.
4. **Caching**
    - If caching is enabled, the response is hashed and stored in a file within the `cache/` directory. Subsequent requests fetch the cached response if available.
5. **Log Rotation**
    - Implements log rotation to ensure logs remain under 10 KB. Older logs are backed up and archived.
6. **System Monitoring**
    - Logs system metrics like memory usage and CPU load during execution. Metrics are appended to `metrics.log`.
7. **Concurrency**
    - Uses Bash background processes (`&`) to handle multiple requests simultaneously.
8. **Output Handling**
    - Displays the response to the user, either from cache or from the server.

b) FlowChart


Start
Input
Parse URL
Is selected HTTP method GET?
No
Yes
Send GET request
Send POST request
Is caching enabled?
No
Yes
Is there a cached response?
No
Output response received from server
Yes
Output response from cache
Save new response in cache
End

Results

Test Cases

Test Case	Input	Expected Output	Result	Analysis
Basic GET request	`./simplefetch.sh GET http://example.com`	Displays website content	Success	Successfully retrieved and displayed the webpage content. Demonstrates proper request formation and handling.
GET with caching	`./simplefetch.sh GET http://example.com -c`	Displays cached response	Success	Cache mechanism worked correctly; subsequent requests fetched data from the cache, reducing response time.
POST request	`./simplefetch.sh POST http://example.com 'key=value'`	Displays server response	Success	POST data successfully included in the request body. Validated content handling and server interaction.
Concurrent requests	Multiple requests with `&`	All responses processed	Success	Concurrent requests showed stable behavior, showcasing effective threading using Bash background processes.
Log rotation	Execute multiple requests	Logs maintained under 10 KB	Success	Log rotation preserved functionality without overflow, maintaining organized and manageable logs.
Invalid URL format	`./simplefetch.sh GET invalid_url`	Displays error message	Success	Error handling correctly identified and reported invalid URL formats, avoiding script crashes.

Performance Metrics

- Response times for requests were consistently logged.
- System memory and CPU usage remained stable even under multiple concurrent requests.

Code Snippets

```
# Function to parse URL and extract host, path, and protocol
parse_url() {
    local url="$1"
    local protocol host path port

    if [[ "$url" =~ ^http://([^/]+)(/.*)?$ ]]; then
        protocol="http"
        host="${BASH_REMATCH[1]}"
        path="${BASH_REMATCH[2]:-/}"
        port=80
    elif [[ "$url" =~ ^https://([^/]+)(/.*)?$ ]]; then
        protocol="https"
        host="${BASH_REMATCH[1]}"
        path="${BASH_REMATCH[2]:-/}"
        port=443
    else
        echo "Invalid URL format. Supported protocols: http, https."
        exit 1
    fi

    echo "$protocol" "$host" "$port" "$path"
}
```

```
# Function to send HTTP/HTTPS GET request
send_get_request() {
    local url="$1"
    local headers_only="$2"
    local output_file="$3"
    local use_cache="$4"
```

```
    local max_redirects=5
    local redirect_count=0

    monitor_system  # Log system metrics before starting

    local start_time=$(date +%s.%N)

    while true; do
        # Use cache if enabled
        if [ "$use_cache" = true ]; then
            if check_cache "$url"; then
                return
            else
                echo "[INFO] Fetching live response for $url."
            fi
        fi

        # Parse URL components
        read -r protocol host port path <<< "$(parse_url "$url")"

        # Construct the request headers
        request="GET $path HTTP/1.1\r\nHost: $host\r\nUser-Agent:
SimpleFetch/1.0\r\nConnection: close\r\n\r\n"

        # Choose transport based on protocol
        if [ "$protocol" = "https" ]; then
            response=$(echo -e "$request" | openssl s_client -quiet
-connect "$host:$port" 2>/dev/null)
        else
            response=$(echo -e "$request" | nc "$host" "$port")
        fi

        # Log the request and response
        echo "[$(date)] - GET $url" >> simplefetch.log
        log_debug "Request Headers: $request"
        rotate_log "simplefetch.log"

        # Separate headers from the body
        headers=$(echo "$response" | sed '/^\r$/q')
```

```
        if [ "$headers_only" = true ]; then
            if [ -n "$output_file" ]; then
                echo "$headers" > "$output_file"
            else
                echo "$headers"
            fi
        else
            if [ -n "$output_file" ]; then
                echo "$response" > "$output_file"
            else
                echo "$response"
            fi

            # Save to cache if enabled
            if [ "$use_cache" = true ]; then
                save_to_cache "$url" "$response"
            fi
        fi

        # Check for redirect (status code 3xx and Location header)
        status_code=$(echo "$headers" | grep -oP "HTTP/1\.[01] \K[0-9]+")
        location=$(echo "$headers" | grep -i "Location:" | awk '{print
$2}' | tr -d '\r')

        if [[ "$status_code" =~ ^3 && $redirect_count -lt $max_redirects
&& -n "$location" ]]; then
            url="$location"  # Update URL to redirect location
            redirect_count=$((redirect_count + 1))
            echo "Redirecting to $url"
        else
            break
        fi
    done

    local end_time=$(date +%s.%N)
    log_metrics "$start_time" "$end_time" "$url"  # Log response time
}
```

Output

```
rabyaan@DESKTOP-FTJ3SOT:/mnt/f/simpleFetch$ ./simplefetch.sh GET https://google.com
HTTP/1.1 301 Moved Permanently
Location: https://www.google.com/
Content-Type: text/html; charset=UTF-8
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce-ptJ1KTUby298GvcmsX405w' 'strict-dynamic
' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/other-hp
Date: Mon, 25 Nov 2024 20:25:06 GMT
Expires: Wed, 25 Dec 2024 20:25:06 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 220
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
Connection: close

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="https://www.google.com/">here</A>.
</BODY></HTML>
Redirecting to https://www.google.com/
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2024 20:25:07 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce-WuMv8aUkFx6lCb9XmQnjMg' 'strict-dynamic
' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/other-hp
Accept-CH: Sec-CH-Prefers-Color-Scheme
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: AEC=AZ6Zc-WZLbLhE6bdvkxSzLrFyv2KPTjKNSxA_cF77Up3OJoo3GlUJ9AmMA; expires=Sat, 24-May-2025 20:25:07 GMT; path=/; domai
n=.google.com; Secure; HttpOnly; SameSite=lax
Set-Cookie: NID=519=pJr5SEjjye7MWO_i9yaQ5iAn8E7Ky7uzzzYm-UMicGmO1Nb_yPoA2IRF7z5X4ZeFpp0N3ApsypCCM2pIQ1BGW78Wgno2Lw_e4Duh3STtXKys
VsLmb4E2GXRvHdiWMqPRP_CBhwnbTzUQ7Cnz88D9rpoCcfvyRaVncoY3mm58L4hn5k81k4U-AFpWyVRijMss7ZEt; expires=Tue, 27-May-2025 20:25:07 GMT;
 path=/; domain=.google.com; HttpOnly
```

```
rabyaan@DESKTOP-FTJ3SOT:/mnt/f/simpleFetch$ ./simplefetch.sh GET https://google.com -I
HTTP/1.1 301 Moved Permanently
Location: https://www.google.com/
Content-Type: text/html; charset=UTF-8
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce-bc_ZEasRJ6f7jv1oOFwvSw' 'strict-dynamic
' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/other-hp
Date: Mon, 25 Nov 2024 20:32:21 GMT
Expires: Wed, 25 Dec 2024 20:32:21 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 220
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
Connection: close

Redirecting to https://www.google.com/
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2024 20:32:22 GMT
```

```
rabyaan@DESKTOP-FTJ3SOT:/mnt/f/simpleFetch$ ./simplefetch.sh GET https://google.com -o output.txt
Redirecting to https://www.google.com/
rabyaan@DESKTOP-FTJ3SOT:/mnt/f/simpleFetch$
```

metrics.log

```
[Tue Nov 26 01:24:50 PKT 2024] - Free Memory: 7274MB, CPU Load:  0.77
[Tue Nov 26 01:24:51 PKT 2024] - Request to https://lms.shu.edu.pk/login/index.php took 1.338060483s
[Tue Nov 26 01:25:06 PKT 2024] - Free Memory: 7268MB, CPU Load:  0.60
[Tue Nov 26 01:25:07 PKT 2024] - Request to https://www.google.com/ took 1.040268279s
[Tue Nov 26 01:26:14 PKT 2024] - Free Memory: 7247MB, CPU Load:  0.18
[Tue Nov 26 01:26:15 PKT 2024] - Request to https://www.google.com/ took .987517605s
[Tue Nov 26 01:32:22 PKT 2024] - Free Memory: 7239MB, CPU Load:  0.00
[Tue Nov 26 01:32:23 PKT 2024] - Request to https://www.google.com/ took 1.340629252s
[Tue Nov 26 01:33:14 PKT 2024] - Free Memory: 7238MB, CPU Load:  0.00
[Tue Nov 26 01:33:15 PKT 2024] - Request to https://www.google.com/ took .948302422s
```

```
rabyaan@DESKTOP-FTJ3SOT:/mnt/f/simpleFetch$ ./simplefetch.sh POST https://eoh8ch6a27b44ee.m.pipedream.net "name=Rabyaan&course=OS"
rabyaan@DESKTOP-FTJ3SOT:/mnt/f/simpleFetch$
```

```
▶ context {18}
▼ event {7}
  ▼ body {2}
      course: OS
      name: Rabyaan
    client_ip: 205.164.137.95
  ▼ headers {4}
      content-length: 32
      content-type: application/json
      host: eoh8ch6a27b44ee.m.pipedream.net
      user-agent: SimpleFetch/1.0
    method: POST
    path: /
  ▼ query {0}
    url: https://eoh8ch6a27b44ee.m.pipedream.net/
```

Conclusion

SimpleFetch successfully combines educational value with practical utility. While not intended to replace advanced tools like `curl`, it demonstrates the power of Bash scripting for OS-level problem-solving and encourages exploration of system resources and capabilities.