Sobre Bash y el análisis de datos

Sebastián Alí Sacasa Céspedes[1]
[1]Universidad de Costa Rica (UCR),
San Pedro de Montes de Oca, San José, 11501-2060, Costa Rica
Corresponding author(s) E-mail(s): sebastian.sacasa@ucr.ac.cr

January 23, 2026

Abstract

Este trabajo presenta un enfoque metodológico para la manipulación de datos y la generación estructurada de archivos a través del paradigma declarativo de Bash scripting, evitando el uso de estructuras de bucles explícitos. El primer problema aborda el filtrado y la transformación de una base de datos a gran escala (`usuarios.csv`) mediante condiciones combinatorias implementadas a través de expresiones regulares y operadores lógicos en `awk`, `grep` y `sed`. El segundo problema involucra la generación controlada de secuencias de números aleatorios, su concatenación en estructuras columnares y el posterior ordenamiento y análisis de tamaños. Se modela el espacio de datos como un espacio topológico discreto, donde cada operación corresponde a un mapeo continuo que preserva invariantes algebraicos. Esta aproximación proporciona un marco teórico unificado para entender la estabilidad estructural y las propiedades de composición de los flujos de datos en scripts de shell.

Keywords: BASH (Bourne Again Shell), análisis de datos, linux, física computacional.

Contents

1 Introducción

En topología algebraica, los espacios topológicos se estudian mediante functores que asignan invariantes algebraicos, como grupos de homología [9]. De manera análoga, un

archivo de datos puede concebirse como un espacio discreto cuyos puntos corresponden a registros, y las operaciones de transformación se interpretan como mapeos entre espacios. Esta analogía permite aplicar conceptos como homeomorfismos, retracciones y preservación de invariantes al diseño de pipelines de datos [3, 5]. La manipulación de datos estructurados se sitúa así en la intersección entre la informática, las matemáticas aplicadas y la física estadística [4, 14]. Mientras que los lenguajes de programación de alto nivel suelen proveer estructuras incorporadas para el manejo de datos, la shell de Unix ofrece un paradigma complementario: orientado al flujo, composicional y profundamente enraizado en la filosofía de la modularidad y la transformación textual [3, 5].

Este trabajo explora el uso de Bash y sus utilidades para resolver dos clases de problemas: (1) el filtrado y la extensión de conjuntos de datos estructurados bajo restricciones lógicas complejas, y (2) la generación, combinación y análisis de archivos que contienen datos numéricos aleatorios [1]. En el primer problema se aprovechan herramientas como `awk` y `sed` para implementar selecciones basadas en predicados y extensiones estructurales sobre archivos CSV, tratando cada comando no como una mera instrucción, sino como una proposición lógica que opera sobre líneas de texto interpretadas como tuplas. Este enfoque refleja conceptos de la álgebra relacional y de la teoría de modelos finitos, donde los datos se filtran mediante condiciones conjuntivas y disyuntivas [8, 3].

El segundo problema introduce una canalización para la generación de números aleatorios a partir de un script externo, la combinación de múltiples archivos en columnas mediante `paste` y su ordenamiento por columnas específicas mediante `sort` [6, 12]. Este proceso ejemplifica cómo la aleatoriedad puede ser contenida y organizada en estructuras deterministas, un tema que también se encuentra en la mecánica estadística y en las simulaciones de Monte Carlo [11, 4]. En ambos problemas subyace la idea de que las transformaciones de datos pueden entenderse como mapeos continuos en un espacio topológico discreto: cada operación (filtrar, ordenar o concatenar) preserva ciertos invariantes mientras altera otros, de manera análoga a cómo los morfismos en topología algebraica preservan propiedades homológicas [9]. Esta perspectiva eleva el scripting en shell de una tarea meramente procedimental a un marco conceptual para comprender la organización de los datos.

Las secciones siguientes presentan los problemas y sus soluciones en detalle, seguidas de una discusión de sus fundamentos lógicos y matemáticos. Referencias a la física computacional, la teoría de números y la regularización topológica proveen un contexto más amplio para los métodos empleados [10, 13, 2, 14]. Este enfoque interdisciplinario integra topología, álgebra y computación, ofreciendo un marco teórico que sustenta la eficiencia, composicionalidad y verificabilidad de los pipelines de datos en entornos Unix [7, 8, 4].

2 Problema I: Filtrado y Extensión de Datos como Mapeos Continuos

Sea X el espacio de todas las líneas del archivo `usuarios.csv`, dotado de la topología discreta. Cada filtro aplicado mediante `awk`, `grep` o `sed` define un subespacio $Y \subseteq X$ que satisface ciertas condiciones lógicas, y cada operación puede entenderse como un mapeo continuo que preserva invariantes de la estructura de datos.

Para seleccionar los usuarios llamados Brian cuyo correo termina en `.com` y cuyo empleador es Boeing, se utiliza el comando

```
awk -F, '$1 == "Brian" && $3 ~ /\.com$/ && $4 == "Boeing" {print}' usuarios.
    csv
```

Este comando interpreta cada línea como una tupla de cuatro campos separados por comas, correspondientes a nombre, apellido, correo y empleador. La condición `$1 == "Brian"` selecciona registros cuyo primer campo coincide exactamente con "Brian", `$3 ~ /\.com$/` filtra aquellos correos que terminan en `.com`, y `$4 == "Boeing"` asegura que el empleador sea Boeing. La instrucción `print` imprime únicamente los registros que cumplen todas las condiciones. Formalmente, este comando define un mapeo $f : X \longmapsto Y$ donde Y es el subespacio de registros que satisfacen las condiciones, y en términos de topología discreta, f es continuo y preserva la estructura de campos como un invariante.

En un caso más elaborado, para seleccionar usuarios llamados James o Paul, con apellido que comienza en J o S y empleador Ad Astra, se utiliza

```
awk -F, '($1 == "James" || $1 == "Paul") && ($2 ~ /^J/ || $2 ~ /^S/) && $4 ==
    "Ad␣Astra" {print}' usuarios.csv
```

Aquí,

```
($1 == "James" || $1 == "Paul")
```

realiza una disyunción sobre los nombres,

```
($2 ~ /^J/ || $2 ~ /^S/)
```

selecciona apellidos que comienzan con J o S, y

```
$4 == "Ad␣Astra"
```

filtra por empleador. Los paréntesis controlan la precedencia lógica para que las condiciones se apliquen correctamente. Este comando define el mapeo $g : X \to Z$ donde Z es un subespacio determinado por disyunción y conjunción, análogo a un pullback en teoría de categorías.

Para cuantificar usuarios cuyo nombre empieza con la letra M, se emplea

```
grep -c '^M[^,]*,' usuarios.csv > conteo_M.txt
```

La expresión regular `^M[^,]*,` selecciona líneas cuyo primer carácter es M hasta la primera coma, delimitando así el nombre. La opción `-c` devuelve el número de coincidencias en lugar de listarlas, y la salida se redirige a `conteo_M.txt` para preservarla. Este comando puede interpretarse como un functor $h : X \to \mathbb{N}$ que asigna a cada subespacio su cardinalidad, análogo a un functor de homología que asigna invariantes algebraicos a espacios topológicos.

Finalmente, para añadir un campo de número de teléfono a toda la base de datos, se utiliza

```
sed '1s/$/,phone number/; 1!s/$/,5555-5555/' usuarios.csv > usuarios_telefono.
    csv
```

Aquí, `1s/\textdollar/,phone number/` agrega la cabecera del nuevo campo al primer registro, mientras que `1!s/\textdollar/,5555-5555/` añade un valor fijo a todas las demás líneas. La salida se redirige a un nuevo archivo `usuarios_telefono.csv`, preservando la coherencia estructural del original. Este comando define un mapeo $k : X \to X \times \{\text{phone number}\}$ que expande el espacio de datos, es continuo, sobreyectivo y mantiene invariantes la estructura de encabezado.

En conjunto, estos comandos muestran cómo el lenguaje `bash` permite transformar un archivo masivo en un conjunto de proyecciones y recortes que responden a criterios lógicos bien definidos. Cada operación puede entenderse como un mapeo continuo o functorial, lo que convierte la manipulación de datos en un proceso lógico, reproducible y matemáticamente interpretable, en lugar de una simple serie de instrucciones mecánicas.

3 Problema II

El problema en cuestión consiste en manipular un generador externo, `random.sh`, cuya salida produce diez números aleatorios decorados con información adicional. La tarea no se limita a invocar dicho generador, sino que requiere construir sobre él una infraestructura que permita organizar resultados en archivos, concatenarlos de manera ordenada, y manipularlos según criterios de ordenamiento y de tamaño. Cada comando y cada script debe entenderse, más que como una instrucción mecánica, como parte de una arquitectura de razonamiento: se trata de traducir la necesidad de estructurar la aleatoriedad en un conjunto de operaciones textuales precisas, reproducibles y consistentes.

El script `generate_random.sh` define un mapeo $r : \emptyset \to Y$ donde Y es un espacio de N puntos (números aleatorios). Cada punto se genera de forma independiente, análogo a una realización de un proceso estocástico en un espacio de probabilidad.

```
#!/bin/bash

if [ $# -ne 2 ]; then
    echo "Usage: $0 <i> <N>"
    exit 1
fi

i=$1
N=$2
filename="rand_${i}_${N}.dat"

executions=$(( (N + 9) / 10 ))
tempfile=$(mktemp)

for ((j=0; j<executions; j++)); do
    ./random.sh | grep -E '^[0-9]+$' >> "$tempfile"
done

head -n "$N" "$tempfile"> "$filename"
rm "$tempfile"

echo "Generated $filename with $N random numbers."
```

La lógica descansa en dos observaciones esenciales: cada ejecución de `random.sh` ofrece un bloque fijo de diez números, y la cantidad requerida N puede no coincidir con un múltiplo de diez, argumento sostenido por el teorema de partición de Ramanujan [2, 13]. Se calcula el número de ejecuciones necesarias, garantizando un excedente que cubra la demanda. La salida acumulada se canaliza hacia un archivo temporal, filtrando únicamente las líneas numéricas con `grep`. Finalmente, `head` extrae los primeros N números, eliminando el exceso, dejando como único resultado el archivo `rand_i_N.dat`.

3.1 Concatenación de cinco archivos en columnas

El comando

```
paste rand_0_$N.dat rand_1_$N.dat rand_2_$N.dat rand_3_$N.dat rand_4_$N.dat >
    combined_$N.dat
```

define un mapeo $p : Y_0 \times Y_1 \times Y_2 \times Y_3 \times Y_4 \to Z$ donde Z es el espacio producto. Este mapeo es continuo y preserva la estructura de columnas como invariantes. La instrucción `paste` coloca cada archivo en paralelo, formando una tabla donde cada columna corresponde a un origen distinto, respetando la restricción de no usar bucles explícitos.

3.2 Ordenamiento por columnas

El siguiente comando

```
sort -n -k $((i+1)) combined_$N.dat
```

define un mapeo $s : Z \to Z_{\text{ordered}}$ que reordena los puntos según la columna i. La operación es un homeomorfismo que preserva la topología del orden. Para orden descendente se usa:

```
sort -n -r -k $((i+1)) combined_$N.dat
```

El modificador `-n` asegura una comparación numérica, `-k` especifica la columna de referencia y `-r` invierte el orden. Este reacomodo revela patrones que permanecen ocultos en la distribución original.

3.3 Determinación del tamaño de los archivos

Los comandos

```
du -k rand_i_N.dat | cut -f1
du -m rand_i_N.dat | cut -f1
echo "scale=2;␣$(du␣-k␣rand_i_N.dat␣|␣cut␣-f1)␣/␣1024␣/␣1024" | bc
```

definen funtores $d : Y \to \mathbb{R}$ que asignan a cada espacio su tamaño, análogos a funtores de medida en geometría algebraica. `du` provee la información, `cut` aísla el valor, y `bc` realiza cálculos decimales. La evaluación del tamaño se integra así al análisis de manera formal.

4 Discusión

La reinterpretación topológica de las operaciones de datos en Bash permite revelar una estructura algebraica subyacente que proporciona un marco formal para analizar la estabilidad y composicionalidad de los pipelines, ofreciendo una base sólida para el diseño de scripts robustos y verificables [3, 5]. En este contexto, cada línea de un archivo puede considerarse como un punto en un espacio topológico discreto, y las operaciones de filtrado actúan seleccionando subespacios abiertos o cerrados de acuerdo con condiciones lógicas, de manera análoga a los conceptos de topología algebraica [9]. Comandos como awk y sed definen mapeos continuos entre estos espacios, ya que la preimagen de cualquier subconjunto es abierta; esta analogía, aunque trivial en la topología discreta, permite

comprender cómo las transformaciones de datos preservan estructuras esenciales y garantizan consistencia en el flujo de información [3, 5].

Operaciones de agregación, como el conteo o la adición de campos, conservan invariantes fundamentales, tales como la cardinalidad o la estructura de encabezados, de forma similar a la preservación de propiedades homológicas en espacios algebraicos [9]. Asimismo, la concatenación de archivos puede interpretarse como un producto cartesiano de espacios, mientras que los algoritmos de ordenamiento funcionan como homeomorfismos, reordenando elementos sin alterar su estructura esencial, de acuerdo con fundamentos clásicos en teoría de algoritmos [6, 12]. Esta perspectiva no solo proporciona una justificación teórica para operaciones avanzadas de Bash, sino que también se extiende a otras herramientas de manipulación de datos y entornos computacionales, facilitando flujos de trabajo eficientes y verificables [8, 4].

Además, esta aproximación establece conexiones con métodos matemáticos más amplios, como la teoría de números y la generación de secuencias aleatorias [10, 11, 13], ofreciendo un marco interdisciplinario que integra estructuras algebraicas, topológicas y computacionales en la práctica de la programación científica [14, 2]. En conjunto, estas ideas permiten una comprensión profunda de cómo las operaciones discretas de datos pueden interpretarse y optimizarse mediante principios matemáticos, extendiendo su aplicabilidad más allá de la manipulación básica de archivos hacia entornos de simulación y análisis computacional complejos [4].

5 Conclusiones

Se analizaron posibles vías para dar solución a los problemas de la Tarea I. De forma tal que, se establecieron conexiones con topología algebraica, física estadística, análisis de datos y computación; abriendo posibilidades a soluciones interdisciplinarias a problemas clásicos, colaborando a obtener diversas perspectivas sobre un mismo tema, que todavía puede desarrollarse.

References

[1] Brenes, M. (2025). *Tarea 3*. Universidad de Costa Rica.

[2] Andrews, G. E., & Berndt, B. C. (2005). *Ramanujan's Lost Notebook, Part 1*. Springer.

[3] Robbins, A., & Robbins, A. (2013). *Bash Pocket Reference: Help for Power Users and Sys Admins*. O'Reilly Media.

[4] Scherer, P. O. J. (2019). *Computational Physics: Simulation of Classical and Quantum Systems* (3rd ed.). CRC Press.

[5] Newham, C., & Rosenblatt, B. (2005). *Learning the bash Shell: Unix Shell Programming*. O'Reilly Media.

[6] Knuth, D. E. (1997). *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley.

[7] Oetiker, T., Partl, H., Hyna, I., & Schlegl, E. (2021). *The Not So Short Introduction to LaTeX 2e*.

[8] van der Walt, S., Colbert, S. C., & Varoquaux, G. (2014). *The NumPy Array: A Structure for Efficient Numerical Computation*. Computing in Science & Engineering, 13(2), 22–30.

[9] Hatcher, A. (2002). *Algebraic Topology*. Cambridge University Press.

[10] Hardy, G. H., & Wright, E. M. (2008). *An Introduction to the Theory of Numbers*. Oxford University Press.

[11] Kendall, M. G., & Babington-Smith, B. (1946). *Randomness and Random Sampling Numbers*. Journal of the Royal Statistical Society, 109(1), 1–26.

[12] Sedgewick, R., & Wayne, K. (2011). *Algorithms*. Addison-Wesley.

[13] Silverman, R. (2009). *Introduction to Analytic Number Theory*. Springer.

[14] Griffiths, D. J. (2005). *Introduction to Quantum Mechanics*. Pearson Prentice Hall.