Cómo usar `grep` y `sed` desde dentro de Vim.

`grep` se usa para **buscar** (search) y `sed` se usa para **sustituir** o transformar (stream editor). En Vim, tienes herramientas *nativas* que hacen un trabajo similar, y también puedes llamar a las herramientas *externas* (del sistema).

---

1. `grep` en Vim (Buscar 🔍)

La principal forma de usar `grep` en Vim es para buscar un patrón en múltiples archivos sin salir del editor.

El comando `:grep`

Este comando utiliza el programa `grep` de tu sistema y vuelca los resultados en una lista especial llamada **"Quickfix List"**.

Sintaxis básica:

Vim Script
```
:grep [opciones_grep] "patrón" [archivos]
```

Ejemplo 1: Buscar en todos los archivos del directorio Imagina que buscas la función `miFuncion` en todos los archivos `.js` del directorio actual:

Vim Script
```
:grep "miFuncion" *.js
```

Ejemplo 2: Búsqueda recursiva (muy común) Para buscar `miVariable` en todos los archivos del directorio actual y todos sus subdirectorios:

Vim Script
```
:grep -r "miVariable" .
```

(Nota: Algunos `grep` prefieren `-R`. Si usas herramientas modernas como `ripgrep` (rg) o `ag` (The Silver Searcher), Vim a menudo puede configurarse para usarlas en lugar de `grep`, ya que son mucho más rápidas. El comando seguiría siendo `:grep`).

La "Quickfix List": Tu centro de resultados

Después de ejecutar `:grep`, Vim te mostrará el primer resultado. Pero el verdadero poder está en la *Quickfix List*.

1. **Abrir la lista:** Escribe `:copen` o `:cw` (window) para abrir una nueva ventana en la parte inferior con *todos* los resultados.
2. **Navegar los resultados:**
    - Puedes usar el cursor y presionar `Enter` en una línea para saltar a ese archivo y línea.
    - O puedes navegar sin tener la lista abierta con:
        - `:cn` (cnext): Ir al **siguiente** resultado.
        - `:cp` (cprevious): Ir al **anterior** resultado.
        - `:ccl` (cclose): **Cerrar** la ventana de la lista.

Flujo de trabajo:

1. `:grep -r "API_KEY" .` (Busca en todo el proyecto).
2. `:copen` (Ves todos los lugares donde se usa).
3. Navegas por los resultados, haces tus cambios.
4. `:ccl` (Cierras la lista).

---

2. `sed` en Vim (Sustituir 🔁)

Cuando la gente habla de "sed en Vim", el 99% de las veces se refiere al **comando de sustitución nativo de Vim**, que está *inspirado* en `sed`. Es increíblemente potente.

Sintaxis básica:

Vim Script
```
:[rango]s/[buscar]/[reemplazar]/[flags]
```

Desglose del comando

- `:`: Entra al modo de comandos.
- `[rango]`: Define *dónde* buscar.
    - `%`: **En todo el archivo** (el más común).
    - `.`: Solo en la línea actual.
    - `'a,'b`: En un rango de líneas (por ejemplo, `10,20`).
    - `'<,'>`: **Solo en la selección visual** (seleccionas texto con `V` o `v` y luego escribes `:`).
- `s`: El comando `substitute`.

- `/`: Delimitador. Puedes usar casi cualquier carácter, lo cual es útil si buscas una URL o una ruta:
    - `:%s#/home/usuario#/var/www#g` (usa `#` como delimitador).
- `[buscar]`: El texto o patrón (expresión regular) a buscar.
- `[reemplazar]`: El texto con el que se reemplazará.
- `[flags]`: Modificadores de la operación.
    - `g`: **Global**. Sin esto, solo reemplaza la *primera* ocurrencia en cada línea. Con `g`, reemplaza *todas* las ocurrencias en cada línea.
    - `c`: **Confirmar**. Vim te preguntará (y/n/a/q...) antes de cada cambio. ¡Súper útil!
    - `i`: **Ignorar mayúsculas/minúsculas** (case-insensitive).

Ejemplos prácticos de sustitución

Ejemplo 1: Reemplazo simple en todo el archivo Cambiar todas las instancias de "viejo" por "nuevo" en todo el documento.

Vim Script
```
:%s/viejo/nuevo/g
```

Ejemplo 2: Reemplazo con confirmación Cambiar "Error" por "WARNING", pero preguntando cada vez.

Vim Script
```
:%s/Error/WARNING/gc
```

Ejemplo 3: Reemplazo solo en un bloque visual

1. Entra en modo Visual (`v` o `V`).
2. Selecciona las líneas que quieres afectar.
3. Escribe `:`. Vim automáticamente pondrá el rango `:'<,'>`.

Completa el comando:
Vim Script
```
:'<,'>s/foo/bar/g
```

4. (Esto solo afectará al bloque que seleccionaste).

Ejemplo 4: Usando Expresiones Regulares Poner comillas alrededor de todos los números al inicio de una línea:

Vim Script
```
:%s/^[0-9]\+/"&"/g
```

- `^`: Inicio de línea.

- `[0-9]\+`: Uno o más dígitos.
- `"&"`: El `&` en la parte de reemplazo significa "el texto completo que coincidió", así que pone el número encontrado entre comillas.

---

3. Usando `sed` y `grep` *Externos* (Filtrado 🔩)

También puedes usar `sed` y `grep` *externos* para **filtrar el contenido de tu buffer actual**. Esto es menos común, pero útil.

Se hace con el comando `!`.

Sintaxis:

Vim Script
:[rango]![comando]

Ejemplo 1: Filtrar con `grep` Imagina que tienes un archivo de log gigante y *solo* quieres quedarte con las líneas que contienen "ERROR".

1. `:%!grep 'ERROR'`

Al hacer esto, **el contenido de tu archivo será reemplazado** por la salida del comando. Es decir, tu archivo ahora solo contendrá las líneas con "ERROR". (¡Puedes deshacer con `u`!).

Ejemplo 2: Filtrar con `sed` Quieres usar un script de `sed` externo para numerar todas las líneas (un ejemplo tonto, ya que Vim puede hacerlo, pero ilustra el punto).

1. `:%!sed '=' | sed 'N;s/\n/\t/'`

Esto pasa el buffer entero al comando `sed`, y la salida de `sed` reemplaza el buffer. Generalmente, es más rápido y seguro usar el `:s` nativo de Vim, pero esta opción existe para scripts de `sed` complejos.

Resumen del Flujo de Trabajo 🏆

- **Para buscar en el proyecto:** Usa `:grep -r "patron" .` y navega con `:copen`, `:cn`, `:cp`.
- **Para reemplazar en el archivo:** Usa `:%s/buscar/reemplazar/g` (y añade `c` para confirmar).