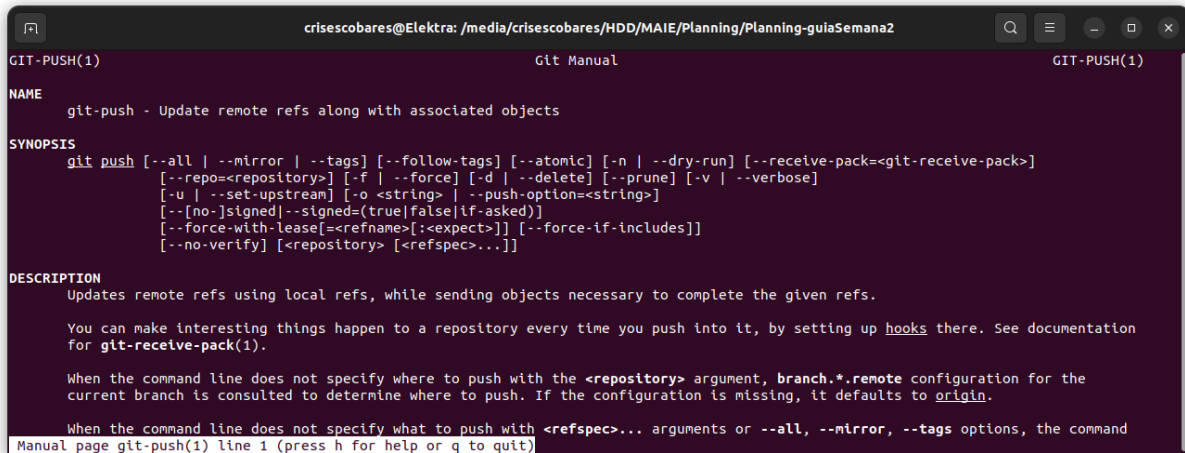


GIT Y GITHUB

Slide 6

1) git help push



```
GIT-PUSH(1)                                Git Manual                                GIT-PUSH(1)

NAME
  git-push - Update remote refs along with associated objects

SYNOPSIS
  git push [--all | --mirror | --tags] [--follow-tags] [--atomic] [-n | --dry-run] [--receive-pack=<git-receive-pack>]
    [--repo=<repository>] [-f | --force] [-d | --delete] [--prune] [-v | --verbose]
    [-u | --set-upstream] [-o <string> | --push-option=<string>]
    [--[no-]signed|--signed=(true|false|if-asked)]
    [--force-with-lease[=<refname>[:<expect>]]] [--force-if-includes]
    [--no-verify] [<repository> [<refspec>...]]

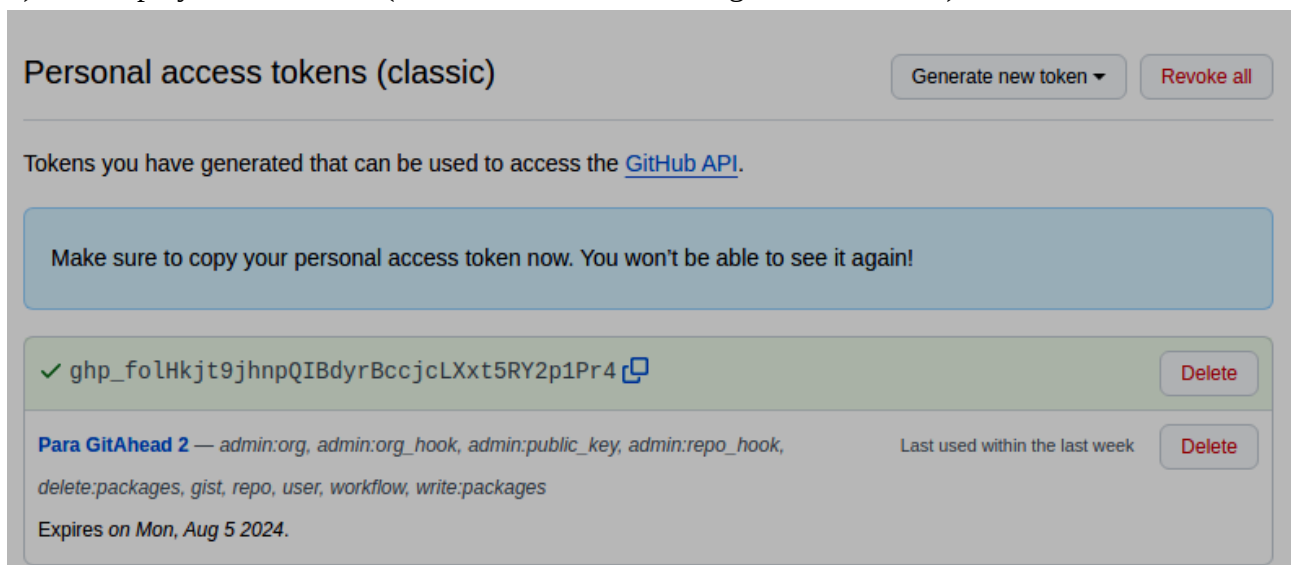
DESCRIPTION
  Updates remote refs using local refs, while sending objects necessary to complete the given refs.

  You can make interesting things happen to a repository every time you push into it, by setting up hooks there. See documentation for git-receive-pack\(1\).

  When the command line does not specify where to push with the <repository> argument, branch.*.remote configuration for the current branch is consulted to determine where to push. If the configuration is missing, it defaults to origin.

  When the command line does not specify what to push with <refspec>... arguments or --all, --mirror, --tags options, the command
Manual page git-push(1) line 1 (press h for help or q to quit)
```

2) Creo repo y token GitHub (el token fue eliminado luego de la creación)



Personal access tokens (classic) Generate new token ▼ Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

✓ 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 Delete

Para GitAhead 2 — admin:org, admin:org_hook, admin:public_key, admin:repo_hook, Last used within the last week Delete
delete:packages, gist, repo, user, workflow, write:packages
Expires on Mon, Aug 5 2024.

Slide 10

1-5) Se hace git revert al commit marcado, que es el anterior al creado (Que en este caso coincide con el estado actual del repo en GitHub)

```
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2

commit e5047d96ec67efafac6ce1e36e8de7e5bbe76356 (HEAD -> main)
Author: cristhianescobares@gmail.com <cristhianescobares@gmail.com>
Date: Fri May 17 15:34:37 2024 -0300

    S10e1-4: trabajado con el método dir de python

commit 62d11806c6921a27ef58e810c495348b05f771e2 (origin/main, origin/HEAD)
Author: cristhianescobares@gmail.com <cristhianescobares@gmail.com>
Date: Fri May 17 15:31:14 2024 -0300

    S6e1-2: Update notación de commits + update README

commit 3313ed8f4b412f125e14623dd6ea2155c7c683b7
Author: cristhianescobares@gmail.com <cristhianescobares@gmail.com>
Date: Fri May 17 15:23:24 2024 -0300

    Completado eje1 y eje2

commit 6b47387ee3f5afc35ad742244412668e35528205
Author: cristhianescobares@gmail.com <cristhianescobares@gmail.com>
Date: Fri May 17 15:13:14 2024 -0300
```

El comando git log nos devuelve el historial de los commits, en HEAD el estado actual de los archivos, en origin/main el estado actual del repo en GitHub. Al lado del comit el id del commit, y abajo información básica de quién realizó el commit, la fecha, y el nombre del commit

----- Slide 12 -----

1-5)

```
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs

Escribiendo objetos: 100% (4/4), 355 bytes | 355.00 KiB/s, listo.
Total 4 (delta 1), reusados 0 (delta 0), pack-reusados 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Escobares-Cristhian/Planning-guiaSemana2
4180bc9..c0efd93  main -> main
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ git branch feature_1
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ git branch feature_2
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ git branch
  feature_1
  feature_2
* main
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$
```

6-9) Al volver a la rama main, no están los cambios realizados en file1.txt dentro de la rama feature_1

```
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs

crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ cat file1.txt
línea1
línea2
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ git commit -m "S12e5-8: file1 y file2 mod
ficados"
[feature_1 8d95c3e] S12e5-8: file1 y file2 modificados
2 files changed, 2 insertions(+)
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ git checkout main
Cambiado a rama 'main'
Tu rama está actualizada con 'origin/main'.
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ cat file1.txt
línea1
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$
```

10-16) Se realizan cambios en ramas y luego un merge:

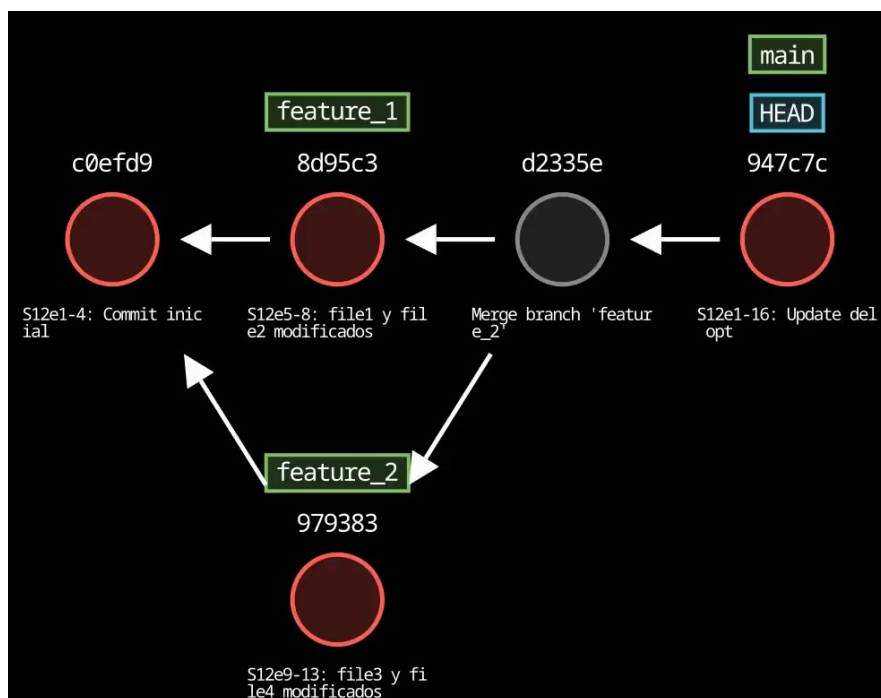
```
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs
Cambiado a rama 'main'
Tu rama está actualizada con 'origin/main'.
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ git merge feature_1
Actualizando c0efd93..8d95c3e
Fast-forward
 prueba_branchs/file1.txt | 1 +
 prueba_branchs/file2.txt | 1 +
 2 files changed, 2 insertions(+)
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ git merge feature_2
Merge made by the 'ort' strategy.
 prueba_branchs/file3.txt | 1 +
 prueba_branchs/file4.txt | 1 +
 2 files changed, 2 insertions(+)
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ cat file*
```

En los archivos file se notan los cambios realizados en todas las ramas:

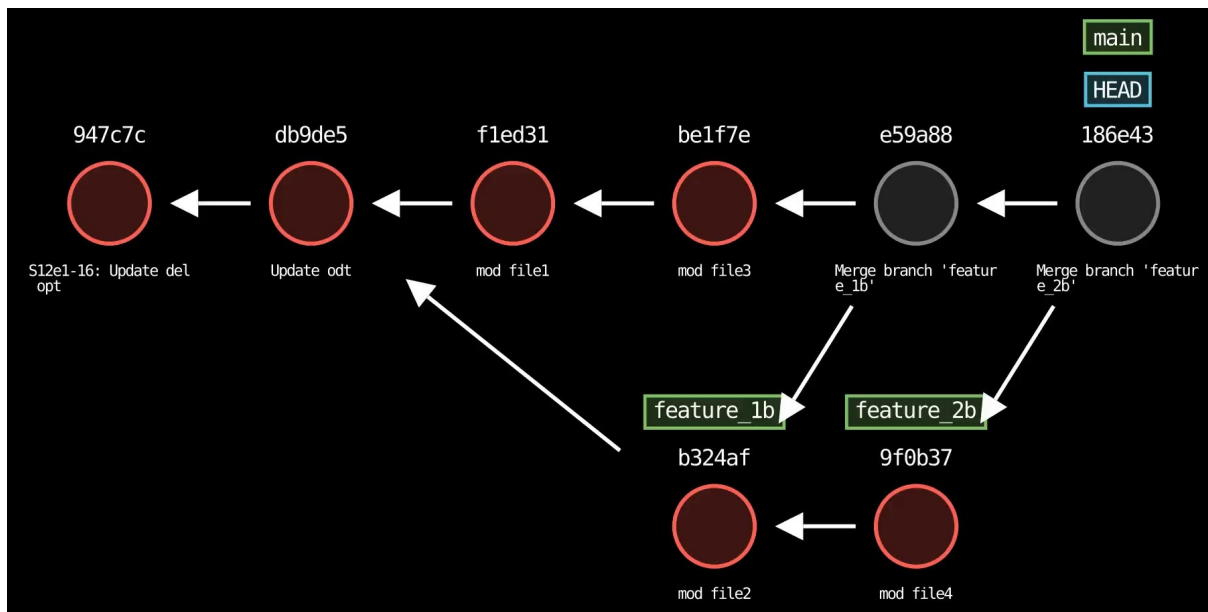
```
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs
linea2
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ cat file1.txt
linea1
linea2
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ cat file2.txt
linea1
linea2
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ cat file3.txt
linea1
linea2
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$ cat file4.txt
linea1
linea2
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/prueba_branchs$
```

----- Slide 13 -----

1) Utilizando la herramienta “git sim” se realiza el gráfico de los commits. Como le feature_1 se hizo merge con main mediante “Fast-Forward” es como si la rama no se hubiera creado porque no se modificó el main durante la vida de la rama feature_1

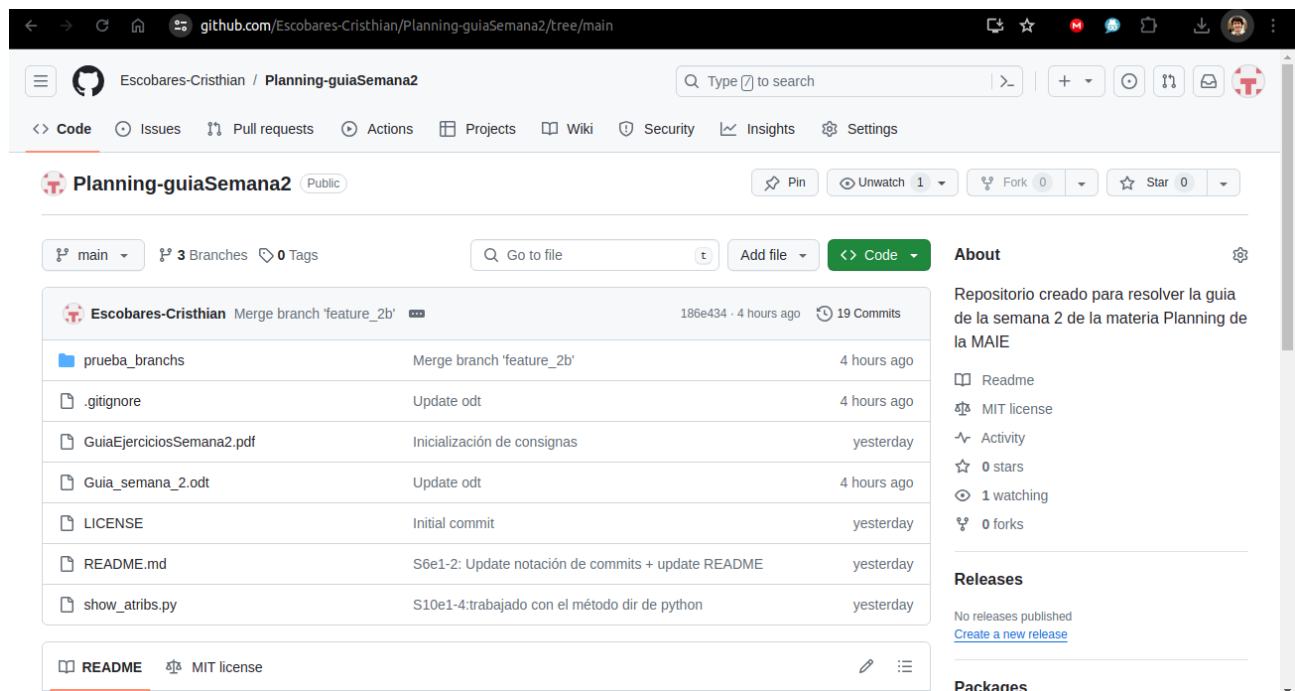


2) Además de lo pedido, se modificó file1.txt y file2.txt en el main para que no se produjera un “Fast-Forward”, ya que al hacer sólo lo que indica la consigna, se obtendrían dos merge “Fast-Forward”.



Slide 15

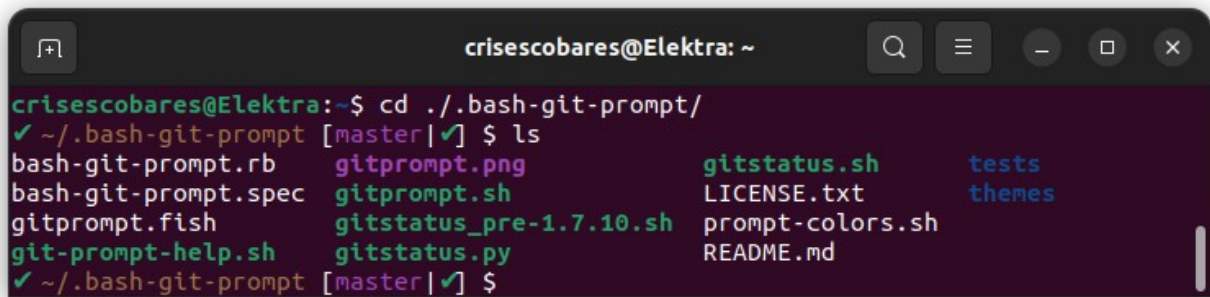
Git push:



Git clone:

Se elige clonar el siguiente repo: <https://github.com/magicmonty/bash-git-prompt> que modifica la terminal de linux para indicar en qué rama de git se encuentra, e información básica sobre cuántos archivos se han modificado, cuántos están en stage, entre otras cosas.

Se ejecuta “git clone https://github.com/magicmonty/bash-git-prompt.git ~/.bash-git-prompt –depth=1” y se mueve a la carpeta para ver su contenido:



```
crisescobares@Elektra: ~  
crisescobares@Elektra:~$ cd ~/.bash-git-prompt/  
✓ ~/.bash-git-prompt [master|✓] $ ls  
bash-git-prompt.rb      gitprompt.png           gitstatus.sh           tests  
bash-git-prompt.spec    gitprompt.sh            LICENSE.txt            themes  
gitprompt.fish          gitstatus_pre-1.7.10.sh prompt-colors.sh  
git-prompt-help.sh      gitstatus.py            README.md  
✓ ~/.bash-git-prompt [master|✓] $
```

-Sistemas Operativos e introducción a la programación-

1) *¿Cómo se define un sistema operativo? ¿Existe una definición concreta o su definición está dada en base a aspectos?*

Un sistema operativo se define como el software principal de una computadora que actúa como intermediario entre el hardware y los programas de aplicación, facilitando la interacción entre el usuario y la máquina. Sus aspectos clave incluyen la interfaz de usuario, la gestión de recursos, el control de procesos y la abstracción de hardware para permitir el funcionamiento eficiente del sistema informático. Por lo tanto, su definición está dada en base a aspectos.

2) *¿Cual es la diferencia entre un proceso y un programa?*

Un programa es un conjunto de instrucciones que están en el disco de almacenamiento, mientras que un proceso es la ejecución activa de un programa en la memoria RAM de la computadora.

3) *¿Con qué comando puede listar los programas instalados en la PC?*

Puede verse con “ls -l /bin”

4) *Listar los procesos de sistema. Hacer un print de pantalla. Observar el estado de algún par de procesos elegidos al azar. ¿Observa que cambian de estado? ¿Cómo podría forzar a que esto ocurra? Lograr ver el cambio de estados es una acción que involucra que la CPU está trabajando de forma más intensiva sobre el proceso, o bien, el proceso tiene más tiempo de CPU.*

Se puede hacer con “ps -aux” o “htop”

```

crisescobares@Elektra: ~
0[|] 3.8% 4[|] 1.3%
1[|] 0.6% 5[|] 0.6%
2[|] 4.5% 6[|] 0.6%
3[|] 0.0% 7[|] 0.6%
Mem[|] 3.95G/7.65G Tasks: 176, 798 thr; 1 running
Swp[|] 1.19G/24.0G Load average: 0.76 0.38 0.47
Uptime: 1 day, 09:17:36

  PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
41746 crisescob  20    0 14868   5760  3584 R   3.2  0.1  0:01.12 htop
1705  crisescob  20    0 5057M  193M  67220 S   2.6  2.5 46:42.25 /usr/bin/gnome-shell
2548  crisescob  20    0 33.1G  94240 71440 S   1.3  1.2 3:06.84 /opt/google/chrome/chrome --type=utility --utility-sub-type=netw
2558  crisescob  20    0 33.1G  51356 42068 S   0.6  0.6 0:15.89 /opt/google/chrome/chrome --type=utility --utility-sub-type=stor
2694  crisescob  20    0 1131G  87032 80904 S   0.6  1.1 0:00.87 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid
27884 crisescob  20    0 559M  52192 36040 S   0.6  0.7 1:05.55 /usr/libexec/gnome-terminal-server
1    root      20    0 164M  11196  7552 S   0.0  0.1 0:03.79 /sbin/init splash
272  root      19   -1 268M  117M  116M S   0.0  1.5 0:28.71 /lib/systemd/systemd-journald
329  root      20    0 27128  6392  4420 S   0.0  0.1 0:00.76 /lib/systemd/systemd-udev
592  systemd-o 20    0 14836  6228  5912 S   0.0  0.1 1:21.61 /lib/systemd/systemd-oomd
593  systemd-r 20    0 26596 10000  8792 S   0.0  0.1 0:08.58 /lib/systemd/systemd-resolved
594  systemd-t 20    0 89388  6660  6372 S   0.0  0.1 0:01.70 /lib/systemd/systemd-timesyncd
636  systemd-t 20    0 89388  6660  6372 S   0.0  0.1 0:00.94 /lib/systemd/systemd-timesyncd
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit

```

En htop, se puede aumentar la prioridad con F8 (Nice +) o disminuirla con F7 (Nice -)

5) Veamos la asignación de recursos de algunos procesos

a) Abrir gedit y vscode (o elegir cualquiera dos programas en ejecución)

b) `ps aux | grep gedit` (obtenemos el PID de gedit)

PID: 41985

c) `ps aux | grep vscode` (obtenemos el PID de vscode)

PID: 36003

d) Ejecutar `htop -p pid1,pid2`

e) Observar CPU y Memoria asignada a los procesos

```

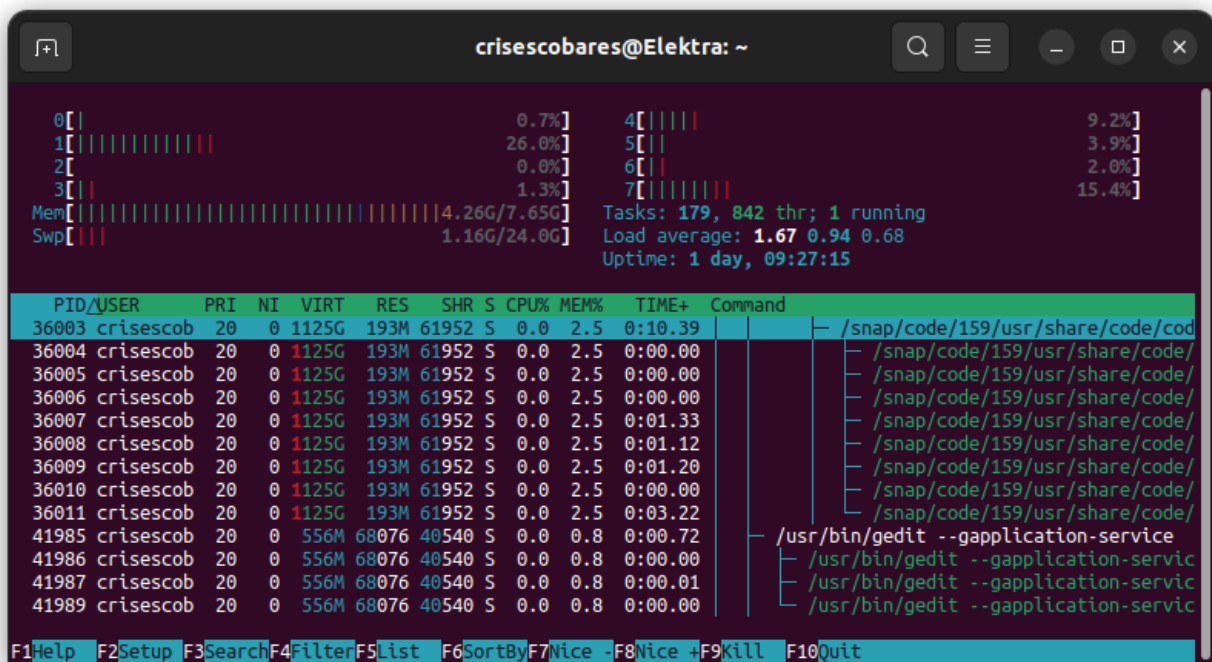
crisescobares@Elektra: ~
0[|] 0.0% 4[|] 1.0%
1[|] 1.0% 5[|] 0.5%
2[|] 10.9% 6[|] 0.0%
3[|] 3.1% 7[|] 0.5%
Mem[|] 4.06G/7.65G Tasks: 176, 800 thr; 1 running
Swp[|] 1.16G/24.0G Load average: 0.48 0.57 0.54
Uptime: 1 day, 09:25:27

  PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
36003 crisescob  20    0 1125G  193M  61952 S   0.0  2.5  0:10.37 /snap/code/159/usr/share/code/code /home/cris
36004 crisescob  20    0 1125G  193M  61952 S   0.0  2.5  0:00.00 /snap/code/159/usr/share/code/code /home/cris
36005 crisescob  20    0 1125G  193M  61952 S   0.0  2.5  0:00.00 /snap/code/159/usr/share/code/code /home/cris
36006 crisescob  20    0 1125G  193M  61952 S   0.0  2.5  0:00.00 /snap/code/159/usr/share/code/code /home/cris
36007 crisescob  20    0 1125G  193M  61952 S   0.0  2.5  0:01.33 /snap/code/159/usr/share/code/code /home/cris
36008 crisescob  20    0 1125G  193M  61952 S   0.0  2.5  0:01.12 /snap/code/159/usr/share/code/code /home/cris
36009 crisescob  20    0 1125G  193M  61952 S   0.0  2.5  0:01.20 /snap/code/159/usr/share/code/code /home/cris
36010 crisescob  20    0 1125G  193M  61952 S   0.0  2.5  0:00.00 /snap/code/159/usr/share/code/code /home/cris
36011 crisescob  20    0 1125G  193M  61952 S   0.0  2.5  0:03.22 /snap/code/159/usr/share/code/code /home/cris
41985 crisescob  20    0 556M  68076 40540 S   0.0  0.8  0:00.66 /usr/bin/gedit --gapplication-service
41986 crisescob  20    0 556M  68076 40540 S   0.0  0.8  0:00.00 /usr/bin/gedit --gapplication-service
41987 crisescob  20    0 556M  68076 40540 S   0.0  0.8  0:00.01 /usr/bin/gedit --gapplication-service
41989 crisescob  20    0 556M  68076 40540 S   0.0  0.8  0:00.00 /usr/bin/gedit --gapplication-service
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit

```

f) Pregunta: ¿Porqué en la salida de htop, se repiten los nombres de los procesos y con diferentes PIDs? ¿Qué pasa si ejecuto `htop -t -p pid1,pid2` ?

Se repiten porque se muestran los procesos hijos de cada proceso, en htop -t -p se muestra con más claridad de que son procesos hijos:



6) ¿Qué es compilar?

Compilar es el proceso de traducir el código fuente de un programa escrito en un lenguaje de programación (como C, C++, Java, Python, etc.) a un lenguaje de máquina que la computadora pueda entender directamente.

7) ¿Que es un compilador?

Un compilador es un programa que compila un programa escrito por algún usuario.

8) En la división lógica de una RAM, que partes existen que son usadas por los programas escritos en lenguajes tipo C? ¿Que se almacenan en estos espacios?

En la división lógica de una RAM, los programas escritos en lenguajes como C utilizan principalmente dos áreas importantes: el **stack** y el **heap**. El **stack** para almacenar variables locales y datos relacionados con la ejecución de funciones, y el **heap** para variables dinámicas cuyo ciclo de vida debe ser gestionado manualmente por el programador.

9) Definir que es variable local y variable global

Variable local: Se declara dentro de una función y solo es accesible dentro de esa función.

Variable global: Se declara fuera de todas las funciones y es accesible en todo el programa.

10) Definir que es una función recursiva

Una función recursiva es una función que se llama a sí misma. Se utiliza principalmente para resolver un problema dividiéndolo en problemas más pequeños, iguales, y secuenciales. Por ejemplo, calcular el factorial de un entero “n”, este puede expresarse como $n! = n \cdot (n-1)!$, por lo que podemos definir la función factorial “f” como $f(n) = n \cdot f(n-1)$

11) Implementar la función factorial en Python y ejecutarla para diferentes entradas. Muestre print de pantalla con el código y las salidas de las ejecuciones para factorial evaluado en: $n = -1, 5, 5000$ ¿Qué cree que sucede cuando $n = -1$? Ayuda: matemáticamente cuál es el dominio e imagen de la función factorial?

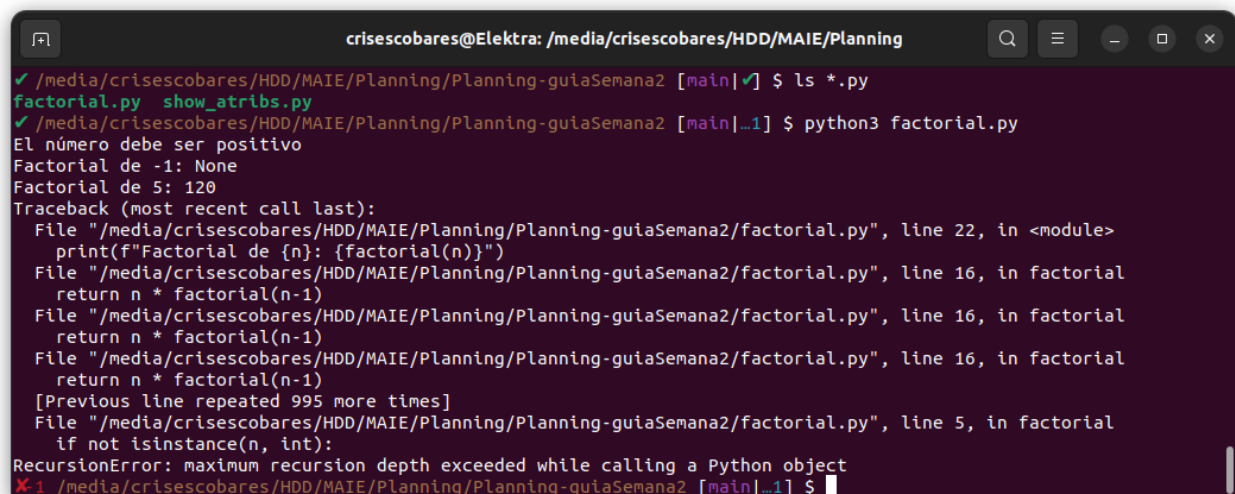
```
def factorial(n):
    # Si no es entero, devolver error
    if not isinstance(n, int):
        raise TypeError("El número debe ser un entero")

    # Si es negativo, devolver error
    if n < 0:
        print("El número debe ser positivo")
        return None

    # Si es menor o igual que 1, el factorial es 1
    elif n <= 1:
        return 1
    else:
        return n * factorial(n-1)

if __name__ == "__main__":
    n_list = [-1, 5, 5000]

    for n in n_list:
        print(f"Factorial de {n}: {factorial(n)}")
```



```
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning
✓ /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2 [main]✓ $ ls *.py
factorial.py show_attrs.py
✓ /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2 [main]_1 $ python3 factorial.py
El número debe ser positivo
Factorial de -1: None
Factorial de 5: 120
Traceback (most recent call last):
  File "/media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/factorial.py", line 22, in <module>
    print(f"Factorial de {n}: {factorial(n)}")
  File "/media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/factorial.py", line 16, in factorial
    return n * factorial(n-1)
  File "/media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/factorial.py", line 16, in factorial
    return n * factorial(n-1)
  File "/media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/factorial.py", line 16, in factorial
    return n * factorial(n-1)
  [Previous line repeated 995 more times]
  File "/media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2/factorial.py", line 5, in factorial
    if not isinstance(n, int):
RecursionError: maximum recursion depth exceeded while calling a Python object
✗ 1 /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2 [main]_1 $
```

Para -1 no está definido el factorial, pues multiplicaría desde -1 hasta -infinito y el valor sería indefinido. Para 5 puede calcularlo exitosamente, pero para 5000 falla, porque llega al límite predefinido máximo de recursión que tiene Python.

12) En lenguajes tipo python o java, como se organiza y gestiona la memoria a diferencia de los lenguajes tipo C? Ayudese con la herramienta <https://pythontutor.com/>

En lenguajes como Python y Java, la gestión de memoria se realiza automáticamente mediante un recolector de basura (garbage collector), mientras que en lenguajes como C, la gestión de memoria es manual y está bajo la responsabilidad del programador.

13) ¿Qué es un lenguaje natural y uno formal?

Lenguaje natural: Evoluciona naturalmente en la comunicación humana, como el español o inglés. Usualmente posee ambigüedades.

Lenguaje formal: Diseñado con reglas precisas para expresar información clara y sin ambigüedades, como el lenguaje de programación C o las gramáticas formales o idiomas inventados, como el Esperanto.

14) ¿Qué elementos mínimos en común tienen dos lenguajes que pertenecen a estos grupos?

Los elementos mínimos en común que tienen los lenguajes de programación incluyen:

Alfabeto: Conjunto de símbolos permitidos.

Gramática: Reglas que definen la sintaxis y estructura.

Semántica: Significado de las construcciones del lenguaje.

15) De una diferencia fundamental que presentan los lenguajes naturales de los formales

La diferencia fundamental radica en la ambigüedad, los lenguajes naturales son usualmente ambiguos, mientras que los formales no lo son, porque se basan en reglas precisas para poder dar órdenes unequivocamente.

16) Suponga que está programando en Python. Quiere crear este diccionario pero algo no está correcto:

```
mi_diccionario = { "compilar": "proceso de traducir código escrito por el usuario a  
código en lenguaje de máquina"  
                  "compilador": "programa para compilar" }
```

¿Qué tipo de error encuentra? en qué parte del proceso desde la escritura del programa por parte del programador hasta la ejecución, se determina este tipo de errores?

Se olvidó la coma entre cada ítem del diccionario, antes de “compilador”. Este error puede encontrarse durante la escritura con la ayuda de correctores de código (como Copilot o similares) o durante la ejecución, ya que devolvería un “Syntax Error” es decir, un error sintáctico que usualmente se detecta antes de comenzar la ejecución.

17) ¿Qué es un algoritmo?

Un algoritmo es un conjunto bien definido, ordenado y finito de instrucciones o pasos que se utilizan para resolver un problema o realizar una tarea específica de manera no ambigua.

18) ¿Qué es un lenguaje de scripting y cuáles conoces?

Un lenguaje de scripting es un tipo especial de lenguaje diseñado para priorizar la rapidez de desarrollo y la flexibilidad, especialmente útil en entornos donde los programadores tienen poca experiencia en otros lenguajes. Ejemplos comunes incluyen Python, Bash, PHP y JavaScript. Estos lenguajes son interpretados, utilizan tipado dinámico para inferir tipos de datos en tiempo de ejecución, ofrecen abstracciones importantes como el manejo de memoria, y son altamente portables, ejecutándose en diferentes sistemas operativos gracias a un intérprete, es decir, no necesitan ser compilados.

19) ¿En qué contextos recomendaría el uso de un lenguaje de scripting vs un lenguaje compilado tipo C?

Los lenguajes de scripting son ideales para la rapidez de desarrollo y la flexibilidad, mientras que los lenguajes compilados son preferibles cuando se requiere un alto rendimiento y eficiencia.

20) ¿Qué características tiene un lenguaje de scripting?

Interpretado: los scripts se ejecutan desde un intérprete, el programador no compila como en C

Tipado dinámico: suele ser una característica y significa que los tipos de datos son inferidos en tiempo de ejecución. En C, tenemos que dar el tipo a cada variable.

Abstracciones: proporcionan abstracciones importantes como el manejo de memoria

Portabilidad: se puede ejecutar el mismo script en diferentes sistemas operativos y arquitecturas, ya que el script depende.

21) ¿Que debe tener en cuenta a la hora de pensar (no de implementar) un algoritmo?

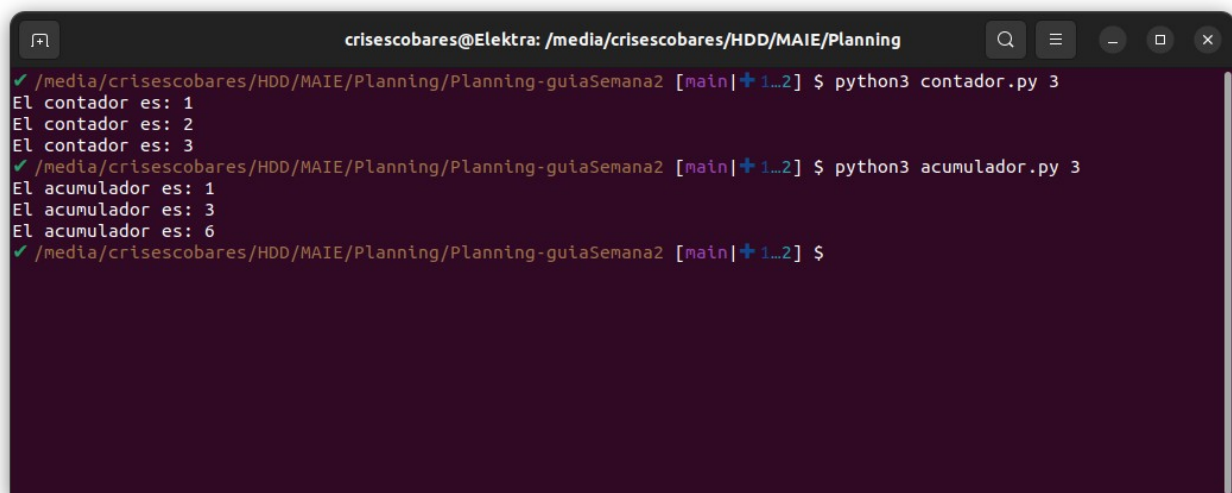
Al pensar en un algoritmo, es importante:

- Entender el problema.
- Dividirlo en partes más pequeñas.
- Diseñar la lógica del algoritmo.
- Considerar su eficiencia.
- Evaluar su flexibilidad y escalabilidad.
- Cómo se generará la documentación del algoritmo para referencia futura.

22) Implementa un ejemplo en Python de un programa que use variables contadores

Implementa un ejemplo en Python de un programa que use variables acumuladores

Para ambos, hacer print de pantalla del código y mostrar la salida de la ejecución.



```
crisescobares@Elektra: /media/crisescobares/HDD/MAIE/Planning
✓ /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2 [main|+1...2] $ python3 contador.py 3
El contador es: 1
El contador es: 2
El contador es: 3
✓ /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2 [main|+1...2] $ python3 acumulador.py 3
El acumulador es: 1
El acumulador es: 3
El acumulador es: 6
✓ /media/crisescobares/HDD/MAIE/Planning/Planning-guiaSemana2 [main|+1...2] $
```

23) Liste los operadores relacionales y lógicos de Python

Operadores relacionales:

- == : Igual que
- != : Diferente de
- > : Mayor que
- < : Menor que
- >= : Mayor o igual que
- <= : Menor o igual que

Operadores lógicos:

- `and` : Devuelve True si ambas condiciones son verdaderas
- `or` : Devuelve True si al menos una de las condiciones es verdadera
- `not` : Devuelve True si la condición es falsa y viceversa
- `|` : Operador OR a nivel de bits (bitwise OR)
- `&` : Operador AND a nivel de bits (bitwise AND)
- `^` : Operador XOR a nivel de bits (bitwise XOR)
- `~` : Operador NOT a nivel de bits (bitwise NOT)

Otros operadores para representaciones binarias (útiles para máscaras de calidad tipo bitmap):

- `<<` : Desplazamiento a la izquierda a nivel de bits (left shift)
- `>>` : Desplazamiento a la derecha a nivel de bits (right shift)

24) Liste los operadores aritméticos de Python

- `+` : Suma
- `-` : Resta
- `*` : Multiplicación
- `/` : División
- `//` : División entera (devuelve la parte entera del cociente)
- `%` : Módulo (devuelve el resto de la división)
- `**` : Exponenciación (eleva un número a la potencia de otro)

25) Explica como funciona un IF y cómo funciona un loop en Python. ¿Cree que funcionan igual en cualquier lenguaje de programación?

Un **if** en Python es una declaración condicional que permite ejecutar un bloque de código si una condición es verdadera. Mientras que un bucle en Python, como el **for** o el **while**, se utiliza para repetir un bloque de código varias veces.

El **if** funciona de la manera:

```
if condición:  
    # código a ejecutar si la condición es verdadera  
else:  
    # código a ejecutar si la condición es falsa
```

El **for loop** funciona de la manera:

```
for variable in secuencia:  
    # código a ejecutar en cada iteración
```

El **while loop** funciona de la manera:

```
while condición:  
    # código a ejecutar mientras la condición sea verdadera
```

No necesariamente funcionan igual en todos los lenguajes de programación, salvo los conceptos básico de las condiciones y los bucles, que son comunes en la mayoría de los lenguajes de programación, pero la sintaxis suele variar de un lenguaje a otro.