E.S. Ingeniería y Tecnología

Ingeniería Informática

Lenguajes y Sistemas Informáticos

Lenguajes y Paradigmas de Programación

Práctica de laboratorio **#4**

1. Iniciar una sesión de trabajo en GNU-Linux.

2. Abra una terminal.

3. Muestre el árbol de directorios de su HOME.(`tree`)

4. Sitúese en el **directorio** de la asignatura "Lenguajes y Paradigmas de Programación" esto es en el directorio *LPP*. (`cd LPP`)

5. Muestre el contenido del directorio actual. (`ls -la`)

6. Cree un nuevo directorio denominado *estructurada* (`mkdir estructurada`). Este será el **directorio de trabajo** durante la realización de esta práctica.

7. Sitúese en el directorio *estructurada*. (`cd estructurada`)

8. Ponga el directorio *estructurada* bajo el control de versiones, es decir, cree un repositorio *git*. (`git init`)

9. Con un editor de textos cree el fichero `.gitignore` en el directorio de trabajo. Este fichero ha de contener la expresión regular que evita almacenar los ficheros acabados en el símbolo ~ denominado suprasegmento, o tilde de la letra ñ.

10. Muestre el estado del repositorio `git` local. (`git status`)

11. Añada todos los ficheros del directorio actual al *índice de git*. (`git add .`)

12. Confirme (*commit*) los cambios. (`git commit -m "Creando el fichero .gitignore"`)

13. Muestre las confirmaciones realizadas en el repositorio hasta el momento.
    (`git log --graph --abbrev-commit --decorate`)

14. Con un editor de textos cree el fichero `README.md` en el directorio de trabajo. Este fichero ha de contener: El nombre del alumno, el número y título de la práctica.

15. Añada todos los ficheros del directorio actual al *índice de git*. (`git add .`)

16. Confirme (*commit*) los cambios. (`git commit -m "Creando el fichero README.md"`)

17. Muestre las confirmaciones realizadas en el repositorio hasta el momento.
    (`git log --oneline --decorate --date=relative --all`)

18. Acepte la tarea asignada en `GitHub Classroom` denominada "Programación estructurada con Ruby". (`https://classroom.github.com/a/iio5VHsw`)

19. Copie la dirección `SSH` del repositorio remoto en *GitHub* de la práctica .
    (`git@github.com:ULL-ESIT-LPP-2223/estructurada-...-aluXXX.git`)

20. En la consola, cree un enlace al repositorio remoto con nombre corto *ghp04*
    (`git remote add ghp04 git@github.com:ULL-ESIT-LPP-2223/estructurada-...-aluXXX.git`)

21. Cambie el nombre de la rama maestra (*master*) a *main*. (`git branch -m main`)

22. Empuje los cambios en el repositorio remoto denominado *ghp04*. (`git push -u ghp04 main`)

23. Muestre las confirmaciones realizadas en el repositorio hasta el momento.
    (`git log --graph --abbrev-commit --decorate --date=relative --all`)

24. Cree una rama *doc*. (`git branch doc`)

25. Muestre la rama activa. (`git branch`)

26. Sitúese en la rama *doc*. (`git checkout doc`)

27. Muestre la rama activa. (`git branch`)

28. Con un editor de textos cree el fichero `respuestas.md` en el directorio de trabajo. Este fichero ha de contener las respuestas a las preguntas que se plantean en esta práctica de laboratorio.

29. Escriba en el fichero `respuestas.md` las respuestas a las preguntas del número 36 al número 68.

30. Cada diez preguntas añada el contenido del directorio al *índice del repositorio git* y confirme los cambios. (`git commit -a -m "Respuesta a las preguntas X-10 a X+10"`)

31. Muestre las confirmaciones realizadas en el repositorio hasta el momento.
    (`git log --graph --abbrev-commit --decorate --date=relative --all`)

32. Al acabar de escribir las respuestas a las preguntas fusionar la rama *doc* en la rama maestra (*main*).
    (`git checkout main; git merge doc`)

33. Empujar los cambios en el repositorio remoto denominado *ghp04*. (`git push -u ghp04 main`)

34. Escriba la dirección HTTP del repositorio de la organización 'ULL-ESIT-LPP-2223' en la tarea habilitada en el campus virtual.

    (`https://github.com/ULL-ESIT-LPP-2223/estructurada-aluXXX.git`)

35. Cierre la sesión.

    Para realizar los siguientes ejercicios utilice el intérprete interactivo *irb* o PRY. Copie **manualmente** los comandos. NO CORTE Y PEGUE desde el fichero PDF, esto puede introducir carácteres extraños e invisibles dependiendo de la codificación que se esté utilizando.

36. ¿Qué diferencia hay entre `"\t\n"` y `'\t\n'`?

37. ¿Cómo funciona `%q`? ¿Qué es `%q{hello world\n}`? ¿Qué es `%q{'a' 'b' 'c'}`?

38. ¿Cómo funciona `%Q`? ¿Qué es `%Q{hello world\n}`? ¿Qué es `%Q{"a" "b" "c"}`?

39. ¿Qué queda en `c`?

```
>> a = 4
=> 4
>> b = 2
=> 2
>> c = <<HERE
0:0:0" --#{a}--
0:0:0" --#{b}--
0:0:0" HERE
```

40. ¿Qué queda en `c`?

```
0:0:0> a = 4
=> 4
0:0:0> b =2
=> 2
0:0:0> c = <<'HERE'
0:0:0' --#{a}--
0:0:0' --#{b}--
0:0:0' HERE
```

41. `s = "hello"`. ¿Cuál es el valor de las siguientes expresiones?

    - `s[0,2]`
    - `s[-1,1]`
    - `s[0,10]`

42. ¿Qué queda en `g`?

```
>> g = "hello"
=> "hello"
>> g << " world"
```

43. ¿Qué queda en `e`?

```
>> e = '.'*3
```

44. ¿Cuál es el resultado?

```
>> a = 1
=> 1
>> "#{a=a+1} "*3
```

45. ¿Qué es esto? `%w[this is a test]`

46. ¿Qué es esto? `%w[\t \n]`

47. ¿Qué es esto? `%W[\t \n]`

48. ¿Qué contiene `nils`? `nils = Array.new(3)`

49. ¿Qué contiene `zeros`? `zeros = Array.new(3, 0)`

50. ¿Qué queda en `b`?

```
>> x = [[1,2],[3,4]]
=> [[1, 2], [3, 4]]
>> b = Array.new(x)
```

51. ¿Qué queda en `c`?

```
 >> c = Array.new(3) { |i| 2*i }
```

52. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> a = ('a'..'e').to_a
=> ["a", "b", "c", "d", "e"]
>> a[1,1]
=>  ???
>> a[-2,2]
=>  ???
>> a[0..2]
=>  ???
>> a[0...1]
=>  ???
>> a[-2..-1]
=>  ???
```

53. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> a
=> ["a", "b", "c", "d", "e"]
>> a[0,2] = %w{A B}
=> ["A", "B"]
>> a
=>  ???
>> a[2..5] = %w{C D E}
=> ["C", "D", "E"]
>> a
=>  ???
>> a[0,0] = [1,2,3]
=> [1, 2, 3]
>> a
=>  ???
>> a[0,2] = []
=> []
>> a
=>  ???
>> a[-1,1] = [ 'Z' ]
=> ["Z"]
>> a
```

```
=>  ???
>> a[-2,2] = nil
=> nil
>> a
=>  ???
```

54. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> a = (1...4).to_a
=> ???
>> a = a + [4, 5]
=> ???
>> a += [[6, 7, 8]]
=> ???
>> a = a + 9
=> ???
```

55. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> x = %w{a b c b a}
=> ???
>> x = x - %w{b c d}
=> ???
```

56. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> z = [0]*8
=> ???
```

57. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> a = []
=> []
>> a << 1
=> ???
>> a << 2 << 3
=> ???
>> a << [4, 5, 6]
=> ???
>> a.concat [7, 8]
=> ???
```

58. ¿Cuál es el resultado de cada una de estas operaciones?

```
>> a = [1, 1, 2, 2, 3, 3, 4]
=> [1, 1, 2, 2, 3, 3, 4]
>> b = [5, 5, 4, 4, 3, 3, 2]
=> [5, 5, 4, 4, 3, 3, 2]
>> c = a | b
=> ???
>> d = b | a
=> ???
>> e = a & b
=> ???
>> f = b & a
=> ???
```

59. ¿Qué resultados dan las siguientes operaciones?

```
>> a = 1..10
=> 1..10
>> a.class
=> Range
>> a.to_a
```

```
=> ???
>> b = 1...10
=> 1....10
>> b.to_a
=> ???
>> b.include? 10
=> ???
>> b.include? 8
=> ???
>> b.step(2) {|x| print "#{x} " }
=> ???
>> 1..3.to_a
=> ???
```

60. ¿Qué resultados dan las siguientes operaciones?

```
>> r = 0...100
=> 0....100
>> r.member? 50
=> ???
>> r.include? 99.9
=> ???
>> r.member? 99.9
=> ???
```

61. ¿Qué resultados dan las siguientes operaciones?

```
>> true.class
=> ???
>> false.class
=> ???
>> puts "hello" if 0
=> ???
>> puts "hello" if nil
=> ???
>> puts "hello" if ""
=> ???
```

62. ¿Qué resultados dan las siguientes operaciones?

```
>> x = :sym
=> :sym
>> x.class
=> ???
>> x == 'sym'
=> ???
>> x == :sym
=> ???
>> z = :'a long symbol'
=> :"a long symbol"
>> z.class
=> ???
>> x == 'sym'.to_sym
=> ???
>> x.to_s == 'sym'
=> ???
```

63. ¿Qué resultados se dan?

```
>> s = "Ruby"
=> "Ruby"
>> t = s
=> ???
```

```
>> t[-1] = ""
=> ???
>> print s
 ???
>> t = "Java"
=> ???
>> print s, t
 ???
```

64. ¿Cuál es el resultado?

```
>> "%d %s" % [3, "rubies"]
=> ???
```

65. ¿Cuáles son los resultados?

```
>> x, y = 4, 5
=> ???
>> z = x > y ? x : y
=> ???
>> x,y,z = [1,2,3]
=> ???
```

66. ¿Qué resultados dan las siguientes operaciones?

```
>> x = { :a => 1, :b => 2 }
=> {:b=>2, :a=>1}
>> x.keys
=> ???
>> x.values
=> ???
>> x[:c] = 3
=> 3
>> x
=> ???
>> x.delete('a')
=> nil
>> x
=> ???
>> x.delete(:a)
=> 1
>> x
=> ???
>> x = { :a => 1, :b => 2, :c => 4 }
=> {:b=>2, :c=>4, :a=>1}
>> x.delete_if { |k,v| v % 2 == 0 }
=> ???
>> x
=> ???
```

67. ¿Qué hace la siguiente sentencia? `counts = Hash.new(0)` ¿Qué diferencia hay con la asignación `counts = {}`?

68. ¿Qué retorna esta expresión regular? `'hello world, hello LPP'.scan /\w+/`