



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 y las respuestas a las preguntas que se plantean en esta práctica de laboratorio.
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 “Primeros pasos con Ruby”.
19. Copie la dirección SSH del repositorio remoto en *GitHub* de la práctica .
(`git@github.com:ULL-ESIT-LPP-1920/ruby-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-1920/ruby-aluXXX.git`)
21. Empuje los cambios en el repositorio remoto denominado *ghp04*. (`git push -u ghp04 master`)
22. Muestre las confirmaciones realizadas en el repositorio hasta el momento.
(`git log --graph --abbrev-commit --decorate --date=relative --all`)
23. Cree una rama *doc*. (`git branch doc`)
24. Muestre la rama activa. (`git branch`)
25. Sitúese en la rama *doc*. (`git checkout doc`)
26. Muestre la rama activa. (`git branch`)
27. Escriba en el fichero `README.md` las respuestas a las preguntas del número 34 al número 66.
28. 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"`)
29. Muestre las confirmaciones realizadas en el repositorio hasta el momento.
(`git log --graph --abbrev-commit --decorate --date=relative --all`)
30. Al acabar de escribir las respuestas a las preguntas fusionar la rama *doc* en la rama *master*.
(`git checkout master; git merge doc`)
31. Empujar los cambios en el repositorio remoto denominado *ghp04*. (`git push -u ghp04 master`)
32. Escriba la dirección HTTP del repositorio de la organización ‘ULL-ESIT-LPP-1920’ en la tarea habilitada en el campus virtual. (`https://github.com/ULL-ESIT-LPP-1920/ruby-aluXXX.git`)
33. 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 caracteres extraños e invisibles dependiendo de la codificación que se esté utilizando.

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

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

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

37. ¿Qué queda en c?

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

38. ¿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
```

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

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

40. ¿Qué queda en g?

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

41. ¿Qué queda en e?

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

42. ¿Cuál es el resultado?

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

43. ¿Qué es esto? %w[this is a test]

44. ¿Qué es esto? %w[\t \n]

45. ¿Qué es esto? %W[\t \n]

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

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

48. ¿Qué queda en b?

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

49. ¿Qué queda en c?

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

50. ¿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]
=> ???
```

51. ¿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
=> ???
```

52. ¿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
=> ???
```

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

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

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

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

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

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

56. ¿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
=> ???
```

57. ¿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
=> ???
```

58. ¿Qué resultados dan las siguientes operaciones?

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

59. ¿Qué resultados dan las siguientes operaciones?

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

60. ¿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'
=> ???
```

61. ¿Qué resultados se dan?

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

62. ¿Cuál es el resultado?

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

63. ¿Cuáles son los resultados?

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

64. ¿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  
=> ???
```

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

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