Universidad de La Laguna
E.T.S. Ingeniería y Tecnología
Sección de Informática
Área: Lenguajes y Sistemas Informáticos

Lenguajes y Paradigmas de Programación
Práctica de laboratorio **#2**

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 la **Carpeta de Proyecto** de la asignatura Lenguajes y Paradigmas de Programación, esto es, en el directorio *LPP* (`cd LPP`).

   Cada equipo ha de elegir un **Coordinador del Equipo de Trabajo**.

   El **Nombre del Equipo de Trabajo** será:

   LPP_j_k

   donde:

   - *j* puede tomar los valores 'M' o 'T', dependiendo del turno: Mañana o Tarde.
   - *k* puede tomar los valores en el rango 1..20, dependiendo del equipo elegido en la consulta.

   En los ejercicios siguientes el coordinador creará la estructura del 'directorio de trabajo del equipo'. Transformará el directorio de trabajo local en un 'repositorio git'. Creará un 'repositorio en *github*' con el nombre del equipo. Dará permisos de escritura en el repositorio de '*github*' a todos los miembros del equipo y a los profesores de prácticas. Los miembro del equipo 'clonarán' el repositorio de *github*. Todos los miembros del equipo, realizarán un cambio y lo incorporarán al repositorio compartido.

   El **Coordinador del Equipo de Trabajo** ha de realizar los ejercicios del 5 al 35. Después ha de realizar unos ejercicios puntuales.

5. Cree un nuevo directorio con el **Nombre del Equipo de Trabajo** (`mkdir`). Este será el **directorio de trabajo** durante la realización de esta práctica.

6. Sitúese en el directorio de trabajo y cree la estructura de directorios que le permita tener subcarpetas para el código y los documentos, es decir:

   - Un subdirectorio *src*
   - Un subdirectorio *docs*

7. Situado en el directorio de trabajo, inicialícelo para que sea un repositorio `git`.

   (`git init`)

8. Compruebe que se crea el directorio `.git`. (`ls -la`)

9. Muestre el estado del repositorio `git` local, esto es, qué ficheros han cambiado, cuáles son nuevos y cuáles han sido borrados. (`git status`)

10. Añada todos los ficheros y subdirectorios del directorio actual al *índice del repositorio git*.
    (`git add .`)

11. Confirme (*commit*) los cambios del índice en el repositorio git local.
    (`git commit -m "Creando el proyecto del trabajo en equipo"`)

12. Muestre las confirmaciones realizadas en el repositorio hasta el momento. (`git log`)

13. Cree el fichero `.gitignore` en el directorio de trabajo. Este fichero ha de contener las expresiones regurales:
    - Para evitar almacenar los ficheros acabados en `~`
    - Para evitar almacenar los ficheros con extensión `.o`
    - Para evitar almacenar los ficheros con extensión `.class`

14. Cree el fichero `README.md` ('`.md`' formato *markdown*) en el directorio de trabajo. Este fichero ha de contener:
    - El nombre del equipo (cabecera de primer nivel).
    - El título de la práctica (texto libre).
    - El nombre del coordinador y el del miembro del equipo (lista).

15. Compruebe el estado del repositorio `git` local. (`git status`)

16. Añada los ficheros `README.md` y `.gitignore` al *índice del repositorio git*.
    (`git add .`)

17. Confirme (*commit*) los cambios del índice en el repositorio git local.
    (`git commit -m "README.md con la descripción del proyecto y el .gitignore"`)

18. Muestre las confirmaciones realizadas en el repositorio hasta el momento. (`git log`)

19. Cree un fichero `prct02.txt` en en el directorio *docs*. Este fichero ha de contener el texto '`Trabajo en equipo con git`'.
    (`echo "Trabajo en equipo con git" > docs/prct02.txt`)

20. Compruebe el estado del repositorio `git` local. (`git status`)

21. Añada el fichero `prct02.txt` al control de versiones.
    (`git add .`)

22. Añada los cambios al *índice del repositorio git* y confírmelos.
    (`git commit -m "Creada la documentación del proyecto"`)

23. Muestre las confirmaciones realizadas en el repositorio hasta el momento. (`git log`)

24. Descargue del Aula Virtual de la asignatura el fichero `cvi.tgz` en el directorio *src* y descomprímalo.
    (`tar -zxvf cvi.tgz`)

25. Compruebe que aparecen en el directorio *src* los siguientes ficheros (`ls -la`).

```
helloWorld.c    - Lenguaje C   / compilado
helloWorld.cc   - Lenguaje C++ / compilado
helloWorld.sh   - Lenguaje Bash / interpretado
helloWorld.py   - Lenguaje Python / interpretado
HelloWorld.java - Lenguaje Java / compilado e interpretado
```

    Cada uno de los ficheros contiene un programa que muestra por pantalla la frase "Hello World".

26. Compruebe el estado del repositorio `git` local. (`git status`)

27. Añada todos los ficheros y subdirectorios del directorio actual al *índice del repositorio git.*
    (`git add .`)

28. Confirme (*commit*) los cambios del índice en el repositorio git local.
    (`git commit -m "Ficheros con el código fuente"`)

29. Muestre las confirmaciones realizadas en el repositorio hasta el momento. (`git log`)

30. Cree un repositorio en *GitHub*

    a) En la barra de usuario, en la esquina superior derecha de la página, haga clik en el icono de "Crear un repositorio nuevo" (*Create a New Repo*).
    b) Introduzca el **Nombre del equipo de trabajo**.
    c) Seleccione que quiere hacer el repositorio público.
    d) **No** seleccione la casilla de crear el fichero README.md.
    e) Pulse el botón para crear el repositorio (*Create repository*)

31. En la consola, cree un enlace al repositorio remoto con nombre corto *ghp02*
    (`git remote add ghp02 git@github.com:XXX/YYY.git`)

32. Empuje los cambios en el repositorio remoto denominado *ghp02.* (`git push -u ghp02 master`)

33. Muestre los repositorios remotos que están definidos. (`git remote -v`)

34. Muestre los detalles del repositorio remoto denominado *ghp02.* (`git remote show ghp02`)

35. Añada como colaboradores al resto de miembros del equipo en el repositorio *GitHub*

    a) Abra en el navegador el repositorio *Github* del trabajo en equipo.
    b) En la el marco de la derecha, seleccione el menú de configuración del repositorio "*Settings*".
    c) En la el marco de la izquierda, seleccione del menú de opciones "*Collaborators*".
    d) En el campo de texto que aparece, introduzca el nombre de usuario de un compañero de equipo: `alu01000YYYYY`.
    e) Pulse el botón para añadir al colaborador (*Add*)
    f) Repita los pasos 35*d*) y 35*e*) para cada miembro del equipo y los profesores de prácticas.

El **Miembro del Equipo** ha de realizar los ejercicios siguientes, excepto aquellos en los que explícitamente se indique que los tiene que realizar el coordinador.

36. Sitúese en el directorio *LPP* y clone el repositorio remoto que ha creado el coordinador del equipo.

    ```
    $git clone git@github.com:alu01000XXX/YYY.git
    ```

37. Compruebe que tiene la misma estructura de directorios y ficheros que ha creado el coordinador del equipo. (`ls -la`)

38. Muestre los detalles del repositorio remoto denominado *origin*. (`git remote show origin`)

39. Muestre los repositorios remotos que están definidos. (`git remote -v`)

40. Cree una rama *desarrollo*. (`git branch desarrollo`)

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

42. Sitúese en la rama *desarrollo*. (`git checkout nombre`)

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

44. Compilación en C:

    a) Muestre el contenido del fichero `helloWorld.c` sin abrirlo (`cat`).
    b) Compile el fichero `helloWorld.c` con el comando `gcc -o helloWorldC helloWorld.c`
    c) Compruebe que aparece en el directorio actual el fichero `helloWorldC` (`ls -la`)
    d) Ejecute el programa que se ha compilado con el comando `./helloWorldC`

45. Compilación en C++:

    a) Muestre el contenido del fichero `helloWorld.cc` sin abrirlo (`cat`)
    b) Compile el fichero `helloWorld.cc` con el comando `g++ -o helloWorldCPP helloWorld.cc`
    c) Compruebe que aparece en el directorio actual el fichero `helloWorldCPP` (`ls -la`)
    d) Ejecute el programa que se ha compilado con el comando `./helloWorldCPP`

46. Interpretación en Bash:

    a) Muestre el contenido del fichero `helloWorld.sh` sin abrirlo (`cat`)
    b) Ejecute el programa con el comando `bash ./helloWorld.sh`

47. Interpretación en Python:

    a) Muestre el contenido del fichero `helloWorld.py` sin abrirlo (`cat`)
    b) Ejecute el programa con el comando `python ./helloWorld.py`

48. Compilación e interpretación en Java:

    a) Muestre el contenido del fichero `HelloWorld.java` sin abrirlo (`cat`)
    b) Compile el fichero `HelloWorld.java` con el comando `javac HelloWorld.java`
    c) Compruebe que aparece en el directorio actual el fichero `HelloWorld.class` (`ls -la`)
    d) Ejecute el programa que se ha generado con el comando `java HelloWorld`

49. ¿Cuál es la diferencia entre compilación e interpretación?

    El coordinador será el responsable de la documentación y ha de escribir la respuesta en el fichero `prct02.txt` creado en el ejercicio 19. Para hacerlo ha de seguir los siguientes pasos:

    a) Cree una rama *documentacion*. (`git branch documentacion`)
    b) Muestre la rama activa. (`git branch`)
    c) Sitúese en la rama *documentacion*. (`git checkout documentacion`)
    d) Muestre la rama activa. (`git branch`)
    e) Abra el fichero '`prct02.txt`' y escriba la respuesta a la pregunta.
    f) Compruebe el estado del repositorio `git` local. (`git status`)
    g) Añada el fichero '`prct02.txt`' al *índice del repositorio git*. (`git add .`)
    h) Confirme los cambios del índice en el repositorio git local.
    (`git commit -m "Diferencia entre compilación e interpretación"`)
    i) Muestre las confirmaciones realizadas en el repositorio hasta el momento. (`git log`)

50. Cree en el directorio *src* un fichero de texto con nombre `helloWorld.rb` que contenga lo siguiente:
    (`puts 'Hello world'`)

51. Añada el fichero `helloWorld.rb` al control de versiones.
    (`git add helloWorld.rb`)

52. Añada los cambios al *índice del repositorio git* y confírmelos.
    (`git commit -m "Hola Mundo en Ruby"`)

53. Ejecute el programa con el comando `ruby ./helloWorld.rb`

54. ¿Qué permisos tiene el fichero `helloWorld.sh`? ¿Se puede ejecutar directamente? (`./helloWorld.sh`)

55. ¿Qué permisos tiene el fichero `helloWorld.rb`? ¿Se puede ejecutar directamente? (`./helloWorld.rb`)

56. ¿Cuál es la principal diferencia entre el contenido del fichero `helloWorld.sh` y `helloWorld.rb`?
    ¿Qué significado tiene la primera línea del fichero `helloWorld.sh`?

    El responsable de la documentación, esto es, el coordinador ha de escribir la respuesta en el fichero `prct02.txt` creado en el ejercicio 19. Para hacerlo ha de seguir los siguientes pasos:

    a) Muestre la rama activa. (`git branch`)
    b) Sitúese en la rama *documentacion*. (`git checkout documentacion`)
    c) Muestre la rama activa. (`git branch`)
    d) Abra el fichero '`prct02.txt`' y escriba la respuesta a la pregunta.
    e) Compruebe el estado del repositorio `git` local. (`git status`)
    f) Añada el fichero '`prct02.txt`' al *índice del repositorio git*. (`git add .`)

g) Confirme los cambios del índice en el repositorio git local.
(`git commit -m "Significado de la línea del shebang "`)

h) Muestre las confirmaciones realizadas en el repositorio hasta el momento. (`git log`)

57. ¿En qué directorio está instalado el intérprete de *Ruby*? (`which ruby`)

58. Modifique el fichero `helloWorld.rb` para que sea ejecutable. Primero edite el fichero y añada la primera línea correspondiente y a continuación establezca los permisos adecuados. (`chmod u+x helloWorld.rb`)

59. Compruebe el estado del repositorio `git` local. (`git status`)

60. Añada los cambios al *índice del repositorio git* y confírmelos.
(`git commit -a -m "Ruby ejecutable"`)

61. Muestre las confirmaciones realizadas en el repositorio hasta el momento. (`git log`)

62. Muestre las confirmaciones realizadas en el repositorio hasta el momento bien formateadas.
(`git log --pretty=oneline`)
Copie el número de identificación de la confirmación "Ficheros con el código fuente"(`Ctrl + C`)

63. Muestre una confirmación concreta mediante su identificador.
(`git show <commit_id>`)

64. Muestre los ficheros que han sido cambiados en una confirmación concreta mediante su identificador.
(`git diff-tree --name-only -r <commit_id>`)

65. Muestre las confirmaciones para el fichero `helloWorld.rb`.
(`git log helloWorld.rb`)

66. Muestre las diferencias de cada una de las confirmaciones para el fichero `helloWorld.rb`.
(`git log -p helloWorld.rb`)

67. Muestre el autor y el identificador de confirmación de cada una de las líneas del fichero `helloWorld.c`. (`git blame helloWorld.c`)

68. Modifique el fichero `helloWorld.rb` para que reciba un argumento desde la línea de comandos y muestre la frase `Hola Mundo` seguida del nombre especificado. (`puts "Hello World #{ARGV[0]}"`)

69. Añada los cambios al *índice del repositorio git* y confírmelos.
(`git add . && git commit -m "Argumentos en la línea de comandos"`)

70. Sitúese en la rama maestra. (`git checkout master`)

71. Fusione la rama de desarrollo y la rama maestra (Avance rápido - *Fast forward*).
(`git merge desarrollo`)

72. Elimine la rama de desarrollo. (`git branch -d desarrollo`)

73. Empuje los cambios en el repositorio remoto denominado *origin*. (`git push -u origin master`)

74. El **coordinador** ha de realizar los siguientes pasos para dejar su repositorio local y el repositorio remoto en su versión final:

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

    b) Sitúese en la rama maestra. (`git checkout master`)

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

    d) Recuperar los datos del repositorio remoto *ghp02*. (`git fetch ghp02`)
    Este comando sólo recupera la información y la pone en tu repositorio local, no la une automáticamente con su trabajo ni modifica aquello en lo que estaba trabajando.

    e) Fusione la rama remota maestra y la rama maestra local (Avance rápido - *Fast forward*).
    (`git merge ghp02/master`)

    f) Fusione la rama de documentación y la rama maestra local (*Ancestro común*).
    (`git merge documentacion`)

    g) Empujar los cambios en el repositorio remoto denominado *ghp02*. (`git push ghp02 master`)

75. Recuperar los datos del repositorio remoto *origin* y fusionarlos automáticamente con la rama master local. (`git pull origin master`)

76. Todos han de escribir la dirección del repositorio que han creado en GitHub en la tarea habilitada en el campus virtual.

77. Cierre la sesión.