Guía de ejercicios: Linux, Git, venvs

1. Instale una distribución de Linux en un *pendrive*, e inicielo. No cree un *live-cd:* Instálelo con el *pendrive* como unidad principal en la que montar el sistema de archivos.
2. Descargue del sitio oficial de Visual Studio Code, el archivo de extensión *.deb.* No lo instale.
3. Presione la combinación de teclas `CTRL+ALT+F1`. Iniciará una sesión en terminal.
4. Verifique si el paquete `zsh` está instalado. Si no está instalado, instálelo, y reinicie la PC (sólo si NO está en modo live-cd).
5. La PC volverá a iniciar en modo gráfico. No cambie al modo consola. Abra un terminal e instale el software `ttyrec`, utilizando `apt`. Cierre la consola, y en una nueva, ejecute el comando `ttyrec.` Comenzará a grabarse la interacción con la terminal.
6. Ingrese su nombre con el comando echo “<nombre>”, reemplazando lo que está entre signos de mayor y menor por su nombre completo. No olvide las comillas.
7. Ejecute el comando `bc`. Es una sencilla calculadora. Para salir, deberá presionar `CTRL+C`.
    1. Realice una división entera (por ejemplo, 20/5).
    2. Realice una división no entera (por ejemplo, 20/4,8). ¿Cuál es el problema?
    3. Revise el manual de `bc` para conocer el parámetro con el que se cargan las bibliotecas matemáticas. Ejecute `bc` con el argumento correspondiente, y realice la última cuenta para verificar su funcionamiento.
8. Ejecute `firefox`. Observe que es posible iniciar aplicaciones gráficas desde la consola.
9. Ejecute `pidgin`, una aplicación de chat. Posiblemente, al contrario de lo que sucede al ejecutar Firefox, la consola en la que se ejecutó no volvió a entregar el poder al usuario mientras la aplicación funcionaba.
    1. Para cancelar un proceso en curso, presione, en la terminal, la combinación de teclas `CTRL+C`.
    2. Ejecute `pidgin &` (note el “ampersand”). Esto hace que el programa se ejecute como un *daemon*, entregando el poder al usuario una vez inicializado.
10. Determine la versión de *Python* que está instalada en su sistema, utilizando el comando `python` con el argumento correspondiente. Para conocer los argumentos posibles, utilice `python -h`. Debería indicar que la versión instalada es de la versión 3.x.
    1. Si la versión instalada es 2.x:
        1. Determine si existe más de una versión de *python* instalada, utilizando el argumento necesario para listar paquetes instalados de `apt list`.
            1. Si existe una versión 3.x, deberá crear un *alias*, mediante el comando `alias`, y modificando el archivo `.bashrc` o `.zshrc` de su directorio personal. El comando para iniciar la versión 3 de python posiblemente sea `python3`. El *alias* deberá ser tal que `python` sea entendido como `python3`.
            2. Si no existe, deberá instalar `python3`, mediante `apt`.
    2. Verifique si el paquete `python3-pip` está instalado, utilizando el comando `grep`.
        1. Si no está instalado, utilice el gestor de paquetes `apt` para instalarlo.
    3. Verifique si el paquete `python3-venv` está instalado, utilizando el comando `grep`.
        1. Si no está instalado, utilice el gestor de paquetes `apt` para instalarlo.

11. Verifique en qué directorio se encuentra, usando el comando `pwd`.
    1. Navegue, utilizando la consola, hasta el directorio /, utilizando `cd`.
    2. Habra el manul de `ls`, con `man ls`. Para salir del manual, presione `q`.
    3. Liste los archivos con `ls -l`. ¿para qué sirve el `-l`?
    4. ¿Qué diferencia hay entre `ls -l` y `ls -lh`?
    5. ¿Cómo distingo una carpeta/directorio de un archivo?
12. Verifique el espacio libre con `df -h`. ¿Aparece más de un punto de montaje? ¿Cuál es el que debe observar?
13. Ejecute el comando `echo $PATH`. ¿Qué es una variable de entorno? ¿Qué es el *PATH*? Observe la variable de entorno *$HOME.*
14. Muestre un calendario en pantalla con `cal`. ¿Con qué comando muestra el calendario de julio de 2014?
15. Muestre la hora del sistema con el comando `date`.
16. Muestre la cantidad de memoria RAM disponible, con el comando `free`. ¿Qué argumento es necesario para que sea fácil de leer?
    1. ¿Qué es el *swap*?
17. Liste los procesos utilizando `top`, `htop` y `ps -aux`. Note que puede utilizar el mouse con `htop`. Si no está instalado, deberá instalarlo utilizando `apt`.
18. Instale `git`, si no está instalado.
19. Utilice el comando `dpkg -i <nombreDelArchivo>` para instalar Visual Studio Code. Recuerde que para instalar paquetes, debe ser superusuario (*root*). Una vez finalizada la instalación, presione la combinación de teclas `CTRL+ALT+F7` para volver al modo gráfico.
    1. Inicie el programa y descargue las extensiones *Bracket Pair Colorizer* y *Python*.
20. Regístrese y cree un repositorio vacío en *Bitbucket*. Recuerde su contraseña.
    1. Clónelo en su computadora. [Opcional: Utilizando llaves SSH].
21. En la carpeta del nuevo repositorio, cree un archivo de prueba de texto, mediante el comando `echo “este texto quedará dentro de un archivo” > archivonuevo.txt.`
    1. Utilice el comando `git status` para verificar el estado del repositorio.
    2. Agregue el nuevo archivo, mediante `git add`.
    3. Verifique nuevamente el estado, mediante `git status`.
    4. Cree una revisión/*commit* con los cambios.
    5. Envíe los cambios al servidor, con `git push`.
22. Cree un nuevo archivo, con otro texto.
    1. Verifique los cambios en la carpeta.
    2. No agregue el archivo a la lista de archivos supervisados. Ejecute un *commit*. Verifique el estado.
23. Modifique el primer archivo, agregue algo al final, mediante el mismo comando, con el operador >> en lugar del operador |.
    1. Muestre el archivo en la terminal, utilizando `cat`, `less` y `vim`, si está instalado. Si no está instalado `vim`, utilice `vi`.
        1. ¿Cómo salir de `vim/vi`?
    2. Utilice el comando `git status` para verificar el estado.

3. Utilice `git diff HEAD` para verificar los cambios que hay entre el *working directory* y el repositorio. Busque, en la ayuda, si es posible poner colores en las diferencias.
4. Utilice `git diff` para verificar los cambios entre el *working directory* y el repositorio. Debería ser igual al comando anterior, debido a que el *index* y el repositorio son iguales.
5. Agregue los cambios al *index*, pero no realice un *commit*. Utilice nuevamente el comando `git diff` para verificar los cambios entre el *working directory* y el *index*. No debería haber cambios. Sin embargo, el comando `git diff HEAD` debería mostrar diferencias, pues los cambios están en el *index* pero no en el repositorio.
6. Realice un *commit* y verifique nuevamente.
7. Envíe los cambios al repositorio.

24. Elimine uno de los archivos utilizando el comando `rm`.
    1. Verifique el estado.
    2. Agregue los cambios.
    3. Utilice `git diff` para verificar los cambios.
    4. Envíelo al servidor.
25. Explore el comando `git log`.
26. Ejecute `git clone` en otra carpeta de su computadora. Esta actuará como una carpeta remota. Usted deberá actualizarla manualmente utilizando `git`.
    1. Agregue un archivo en esta nueva carpeta. Ejecute los pasos necesarios para que queden los cambios en el servidor.
    2. En la primer carpeta, verifique que el archivo agregado no se encuentra.
    3. Realice un `git pull` para traer los cambios desde el servidor. Verifique que ahora el nuevo archivo está disponible.
    4. Realice una modificación al archivo, y envíe los cambios al servidor.
    5. En la segunda carpeta creada (la que da origen a este punto de la guía), realice cambios en el mismo archivo.
        1. Verifique el estado.
        2. Agregue al *index*.
        3. Haga un *commit*.
        4. Traiga los cambios desde el servidor y lea con atención. Verifique el estado de los archivos. Si existe un conflicto irreconciliable automáticamente, deberá interceder.
        5. Haga un *commit* al solucionar los problemas.
        6. Envíe los cambios al servidor.
        7. Ejecute `git log` para ver los cambios.
        8. Ejecute `git log --graph --pretty=oneline –abbrev-commit` para ver los cambios en modo de árbol. ¿Para qué sirve cada uno de los parámetros?
27. Cree un *virtual environment* de Python en la primer carpeta, con el mismo nombre que el repositorio, mediante `python -m venv <nombre>`.
    1. Añada los nuevos archivos a un `.gitignore`, de manera que no se envien al servidor.
    2. Verifique el estado. No debería haber cambios.
    3. Haga *push/pull* en ambas carpetas de trabajo, y verifique que el *virtual environment* no se transmite.

28. Ubíquese en la carpeta donde se creó el *virtual environment* del punto anterior. Liste los archivos de la carpeta `bin`.
    1. Busque el archivo `activate`. Verifique que puede leerlo (con `cat` u otro editor).
    2. Ejecute el comando `which python` para ver qué ejecutable es el que se ejecuta cuando se llama a `python`.
        1. Si la terminal responde que `python` es un *alias*, ejecutar el mismo comando sobre el programa destino (`python3.7`, por ejemplo).
    3. Ejecute el comando `source activate` para activar el *virtual environment*.
    4. Ejecute el comando `which python` para ver qué ejecutable es el que se ejecuta cuando se llama a `python`. ¿Es distinto al anterior?
29. En alguna de las carpetas de trabajo, modifique los archivos de manera tal que resulten irrecuperables, sin destruir el repositorio.
    1. Ejecute el comando `git reset –hard origin/master`, y verifique que todo volvió al estado original.
30. Modifique los archivos de la carpeta de manera tal que resulten reconocibles. Agréguelos, realice un *commit* y envíelos al servidor. Ponga un mensaje reconocible en el *commit*.
    1. Ejecute `git log` para ver el historial de cambios.
    2. Ejecute el comando `git checkout HEAD^1` y observe el directorio de trabajo.
31. Realice varios *commits* (luego de realizar sucesivos cambios) en alguno de los repositorios. Elija uno al que desee volver, observando la salida del comando `git log`. Copie el identificador hexadecimal del *commit* elegido. Intentaremos volver a ese *commit*.
    1. Ejecute el comando `git checkout <identificador>`. Note que antes utilizamos `git checkout HEAD^1`, de estructura similar. Verifique que la carpeta de trabajo haya cambiado. Ahora está parado en un *commit* particular.
    2. Crearemos un *brach* utilizando el comando `git branch <nombre_nuevo_branch>`.
    3. Cambie al nuevo *branch*, utilizando `git checkout <nombre_nuevo_branch>`. Quedará ahora parado en el nuevo *branch*.
    4. Modifique archivos, y realice los *commits* necesarios.
    5. Entre al sitio de B*itbucket* y busque el nuevo *branch* creado.
    6. Ejecute el comando `git branch -a` para listar los *branches* creados. ¿para qué sirve el `-a`?
    7. Vuelva al *branch* original: `git checkout master`.
    8. Verifique que los archivos del *branch* original sean correctos.
    9. Muestre el árbol de *commits* en la consola.
32. Realizaremos el primer *merge*. estando parado en el *branch master*, ejecute `git merge <nombre_nuevo_branch>`.
    1. Si apareció un conflicto, habrá que lidiar con él.
        1. Con `git status` se pueden observar los archivos en conflicto.
    2. Muestre el árbol de *commits* en la consola.
33. Observe su historial de comandos con el comando `history`.
34. Haga una búsqueda inversa de `git checkout` en el historial :
    1. Utilizando `grep` y |.
    2. Utilizando CTRL+R.