Humanytek documentation

27 de junio de 2019

Índice general

humanytek

Productividad es el acto de acercar a la empresa a su meta

CAPÍTULO 1

Cursos

Los cursos listados en esta sección están pensados para que, al finalizarlos, el usuario sea capaz de desarrollar y modificar módulos para Odoo en su versión más reciente.

1.1 Requisitos previos

- Familiaridad con la computación

1.2 Recomendaciones

- Utilizar para todo el curso alguna distribución de Linux (De preferencia basada en Debian)

1.3 Notas

- Todos los requerimientos son cubiertos si se utiliza la imagen de docker humanytekteam/odoo en su tag más reciente.

1.4 Temas

1.4.1 Linux

Duración: 60 min.

En este curso se abordan puntos básicos sobre el manejo de un servidor basado en una imagen de Debian/Ubuntu mediante el uso de la terminal.

Requisitos previos

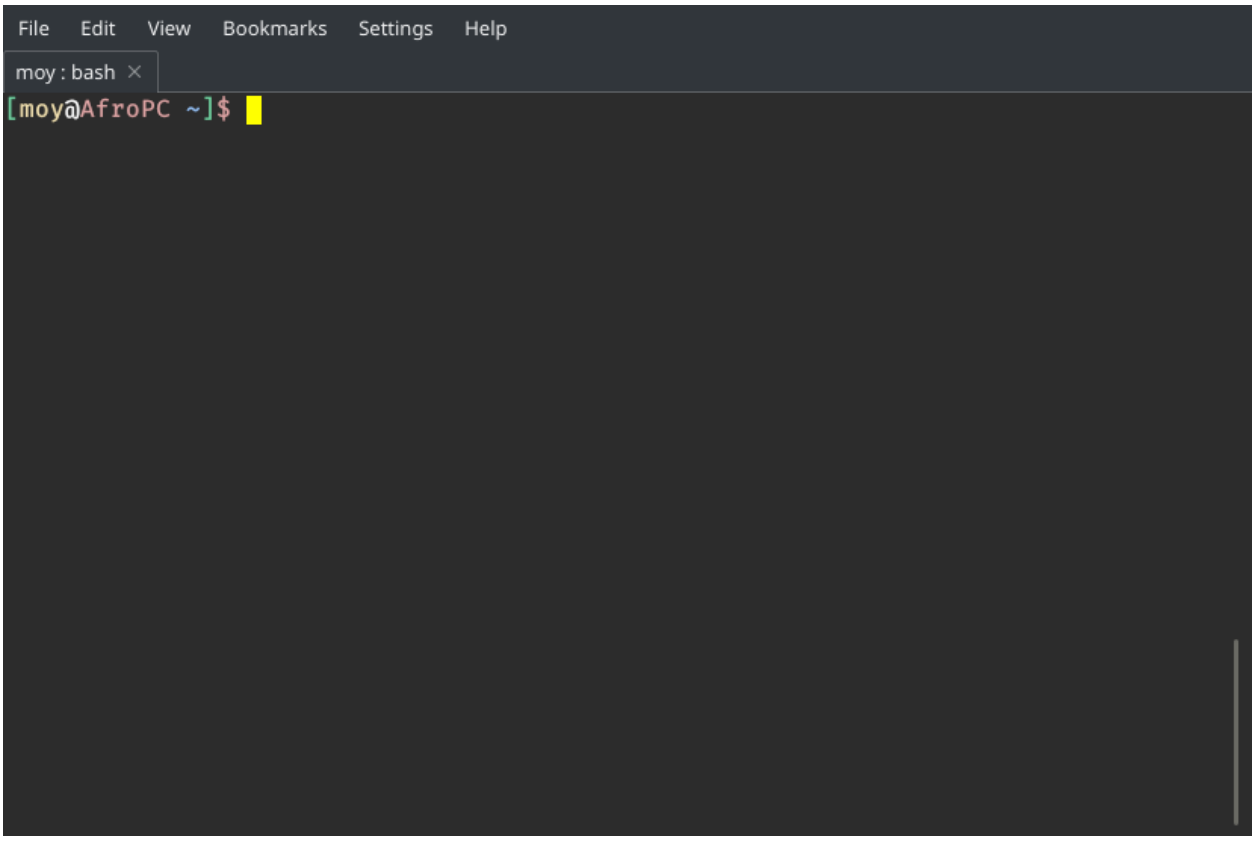
- Computadora (física o virtual) corriendo alguna distribución de GNU/Linux basada en Debian.

Temario

Familiarizándose con la terminal

Lo primero que haremos será familiarizarnos con la terminal. Todo el manejo de los servidores lo haremos mediante esta interfaz.

Lo primero que debemos hacer es notar los elementos en pantalla. (Puede que nuestra terminal no sea idéntica, pero debe tener cierta similitud).



Elementos

1. La parte amarilla que dice *moy* es el nombre de usuario con el que estamos trabajando actualmente (generalmente es el inicio de la cadena).
2. El @ sirve para indicar que un usuario está dentro de alguna máquina (a su izquierda el usuario y a su derecha la máquina).
3. La parte roja que dice *AfroPC* indica el nombre de la máquina en la que estamos trabajando.
4. El símbolo ~ indica la ruta en la que estamos posicionados (Ese símbolo indica que estamos en la ruta *home* del usuario con el que estamos trabajando).
5. El símbolo *$* nos indica que la terminal está lista para recibir ordenes de un usuario no-root.

Navegando en el sistema

Una vez que nos hemos familiarizado con la terminal podemos empezar a movernos por el sistema, para ello es necesario que utilicemos los siguientes comandos.

pwd

Este comando nos indica en que ruta estamos trabajando.

```
[moy@AfroPC ~]$ pwd
/home/moy
```

ls

Este comando nos muestra el contenido dentro de un directorio.

```
[moy@AfroPC ~]$ ls
file1 file2 dir1  dir2
```

mkdir

Con este comando creamos carpetas o directorios.

```
[moy@AfroPC ~]$ ls
file1 file2 dir1  dir2
[moy@AfroPC ~]$ mkdir dir3
[moy@AfroPC ~]$ ls
file1 file2 dir1  dir2  dir3
```

touch

Con este comando podemos crear o actualizar la fecha de modificación de un archivo.

```
[moy@AfroPC ~]$ ls
file1 file2 dir1  dir2  dir3
[moy@AfroPC ~]$ touch file3
[moy@AfroPC ~]$ ls
file1 file2 file3 dir1  dir2  dir3
```

cd

Para cambiar de ruta en la que trabajamos usamos el comando *cd* (change directory).

```
[moy@AfroPC ~]$ pwd
/home/moy
[moy@AfroPC ~]$ cd dir1
[moy@AfroPC ~]$ pwd
/home/moy/dir1
```

mv

Este comando es utilizado para mover archivos o directorios dentro del sistema (también podemos renombrarlos)

```
[moy@AfroPC ~]$ ls
file1 file2 file3 dir1  dir2  dir3
[moy@AfroPC ~]$ mv file1 file_renamed
[moy@AfroPC ~]$ ls
file2 file3 file_renamed  dir1  dir2  dir3
```

rm

Para eliminar un archivo se utiliza el comando *rm*

```
[moy@AfroPC ~]$ ls
file2 file3 file_renamed  dir1  dir2  dir3
[moy@AfroPC ~]$ rm file_renamed
[moy@AfroPC ~]$ ls
file2 file3 dir1  dir2  dir3
```

Manejando archivos

Ya que sabemos como movernos y modificar archivos y ficheros como unidad, ahora veremos como manipular el contenido de los mismos.

Para lograr esto, comúnmente, se utiliza un editor de texto, es muy probable que su distribución venga con *vim* o *nano* como editor de textos en terminal.

vim o *nano*

Con estos editores podemos ver y modificar el contenido de un archivo de texto.

```
[moy@AfroPC ~]$ ls
file2 file3 file_renamed  dir1  dir2  dir3
[moy@AfroPC ~]$ vim file2
```

```
Contenido del archivo <file2>
```

Nota: Para cerrar *vim* presiona *:q* seguido de *intro*. Para *nano* utiliza *c-x*.

cat

Si solo queremos ver el contenido de un archivo (sin editarlo), este comando es muy útil.

```
[moy@AfroPC ~]$ cat file2
Contenido del archivo <file2>
```

tail

Para obtener las ultimas lineas de un archivo (muy útil para ver los últimos eventos en un log) podemos usar este comando.

```
[moy@AfroPC ~]$ tail file2
Contenido del archivo <file2>
```

Nota: Si pasamos el argumento *-f* a *tail*, obtenemos una versión «en vivo» del archivo, de tal modo que vemos los cambios en tiempo real.

grep

Este comando sirve para buscar cadenas dentro de archivos (también puede ser dentro de otras cadenas).

```
[moy@AfroPC ~]$ grep -r archivo
file3:Contenido del archivo <file3>
file2:Contenido del archivo <file2>
```

find

Con este podemos buscar archivos o directorios en base a su nombre. (Podemos usar regex).

```
[moy@AfroPC ~]$ find -name "*file*"
./file2
./file3
./dir1/file54
```

Permisos

Quizá nos hayamos topado con algún archivo o directorio al cual no hemos podido acceder, esto se debe al tema de los permisos dentro del sistema. Cada archivo o directorio le pertenece a un usuario y a un grupo dentro del sistema. Además de esta relación posee atributos que indican lo que el propietario, el grupo al que pertenece y los demás usuarios son capaces de hacer con este objeto.

ls -l

Para ver los permisos de un objeto podemos usar el comando *ls -l* el cual listará los archivos junto con información adicional (entre ella los temas de los permisos).

```
[moy@AfroPC ~]$ ls -l
total 20
drwxr-xr-x 2 moy  moy  4096 May 31 12:50 dir1
drwx--x--x 2 root root 4096 May 31 13:02 dir2
drwxr-xr-x 2 moy  moy  4096 May 31 12:50 dir3
-rw-r--r-- 1 moy  moy    30 May 31 12:35 file2
-rw-r--r-- 1 moy  moy    30 May 31 12:35 file3
-rw-r--r-- 1 moy  moy     0 May 31 12:50 file_renamed
```

Explicación

1. El primer carácter indica la naturaleza del objeto (la *d* es de directorio).
2. **Los siguientes 9 son 3 bloques de 3 elementos cada uno(*r* ead, *w* rite & e *x* ecute).**
    1. A nivel de usuario (u)
    2. A nivel de grupo (g)
    3. A nivel global (o)
3. Las siguientes dos cadenas indican el usuario y el grupo al que pertenecen.

sudo o *su*

Para poder modificar objetos a los que normalmente no tenemos acceso podemos usar el comando *sudo* para otorgarnos poderes de superusuraio de manera temporal o el comando *su* para cambiarnos al usuario *root* (Esto ultimo no es recomendable, pues se corre el riesgo de modificar archivos sin querer).

```
[moy@AfroPC ~]$ ls dir2
ls: cannot open directory 'dir2': Permission denied
[moy@AfroPC ~]$ sudo ls dir2
[sudo] password for moy:
file_secret
```

chown, *chgrp* y *chmod*

Estos comandos son utilizados para modificar los permisos de un objeto. Con *chown* modificamos el propietario de un objeto, con *chgrp* el grupo al que pertenece y con *chmod* los permisos en sí.

```
[moy@AfroPC ~]$ ls -l
total 20
drwxr-xr-x 2 moy  moy  4096 May 31 12:50 dir1
drwx--x--x 2 root root 4096 May 31 13:02 dir2
drwxr-xr-x 2 moy  moy  4096 May 31 12:50 dir3
-rw-r--r-- 1 moy  moy    30 May 31 12:35 file2
-rw-r--r-- 1 moy  moy    30 May 31 12:35 file3
-rw-r--r-- 1 moy  moy     0 May 31 12:50 file_renamed
[moy@AfroPC ~]$ sudo chown moy dir2
[moy@AfroPC ~]$ sudo chgrp moy dir2
[moy@AfroPC ~]$ sudo chmod g+r,o+r dir2
[moy@AfroPC ~]$ ls -l
total 20
drwxr-xr-x 2 moy moy 4096 May 31 12:50 dir1
drwxr-xr-x 2 moy moy 4096 May 31 13:02 dir2
drwxr-xr-x 2 moy moy 4096 May 31 12:50 dir3
-rw-r--r-- 1 moy moy   30 May 31 12:35 file2
-rw-r--r-- 1 moy moy   30 May 31 12:35 file3
-rw-r--r-- 1 moy moy    0 May 31 12:50 file_renamed
```

Más poder

Como es evidente, no podemos abarcar todos los temas y comandos de Linux en este curso, sin embargo este sistema nos ofrece la bondad de poder saber más acerca de sus herramientas sin salirnos del mismo.

man

Con este comando podemos acceder a los *man* uales de los programas dentro del sistema, basta con ejecutar el comando *man APP* (donde *APP* es el programa que queremos explorar) para que se nos despliegue la información relevante acerca de un programa. *Nota:* Algunos programas no incluyen un manpages (paginas de manual), sin embargo podemos probar con el argumento *-h* o *–help* para obtener información adicional del mismo.

whatis

Con este comando podemos obtener una muy breve descripción de que es lo que hace un comando.

apropos

Es común que no recordemos el nombre de algún comando que necesitemos, sin embargo con el comando *apropos*, si le pasamos texto plano como argumentos, podemos obtener una lista de comandos que tengan relación al texto que especificamos.

whereis

En algunas ocasiones dará la necesidad de saber la ubicación del binario de un programa, para ello podemos usar el comando *whereis APP* (donde *APP* es el programa que buscamos) para saber la ruta en donde se encuentran los binarios utilizados por el mismo.

Obtener nuevos programas

Las distribuciones diseñadas para ser usadas en servidores contiene muy pocos paquetes pre-instalados, es muy probable que debamos instalara más dependiendo de nuestras necesidades.

apt-get

Los sistemas basados en Debian cuentan con este gestor de paquetes el cual permite instalar software adicional al sistema.

```
[moy@AfroPC ~]$ apt-get update # Esto actualiza los repositorios utilizados para␣
↪obtener los programas

[moy@AfroPC ~]$ apt-get upgrade # Actualiza los programas instalados a la versión mas␣
↪reciente listada en los repositorios internos.
[moy@AfroPC ~]$ apt-get install python3 # Con `install` instalamos los paquetes
```

Manejo de servidores

Lo que hemos visto aplica para sistemas Linux en general, sin embargo lo más deseado es que esto lo hagamos en un servidor dedicado y no en una computadora personal.

ssh

Para conectarnos a una computadora remota utilizamos el comando *ssh USER@MACHINE* donde *USER* es el usuario dentro de la máquina remota con el cual queremos acceder y *MACHINE* es la dirección de la máquina (puede ser IP o su hostname)

```
[moy@AfroPC ~]$ ssh david@mi_server
[david@mi_server]$
```

crontab

Con este comando podemos automatizar la ejecución de tareas dentro del sistema. Con *crontab -e* abriremos nuestro editor de texto y podemos modificar el archivo para programar eventos periódicos.

systemctl o *service*

Con estos comandos (dependiendo de la distribución puede que no este uno o otro) podemos definir el comportamiento de los demonios (así se llaman los servicios que corren en segundo plano) para realizar cosas como *stop*, *restart*, *start*, *status*.

top

Para conocer que servicios y procesos están siendo ejecutados en el sistema podemos usar el comando *top* el cual nos desplegará una lista con los mismos. Un dato sumamente importante que se nos muestra es el *PID* (process ID).

kill

Si hay algún proceso que queramos terminar de manera instantánea (ojo, puede no activar ciertos mecanismos al momento de forzar el cierre), podemos usar el comando *kill PID*, donde *PID* es el *PID* del proceso a eliminar.

Maestro de la terminal

Por ultimo me gustaría compartir unos «trucos» para ser más eficiente a la hora de hacer uso de la terminal.

Ctrl-C

Con esta combinación de teclas podemos detener la mayoría de los programas que tengan una CLI (command line interface). En caso de que no funcioné prueba con *Ctrl-D*, esto envía una señal del *EOF* (end of file) lo cual suele ser indicador de que se debe terminar la ejecución.

Mouse 3

Dentro de algunas terminales es posible pegar el texto seleccionado (desde cualquier lado) mediante el uso del botón 3 del ratón (generalmente el click de la rueda).

1.4.2 GitHub

Duración: 30 min.

En este curso se abordarán temas relacionados al gestor de versiones *git* y la integración con la plataforma github.

Requisitos previos

- Cliente *git*
- Cuenta en github

Temario

Qué es un repositorio?

Un repositorio es una carpeta o directorio en la cual se lleva el control y seguimiento de algún proyecto.

Con los gestores de versiones (como *git*) podemos llevar un control detallado de los cambios que han sufrido los archivos dentro del repositorio y con esto revisar versiones anteriores de los mismos (esto es muy útil para detectar la inyección de nuevos errores en el código).

Además de esto, el uso de control de versiones nos facilitan el trabajo colaborativo puesto que podemos realizar un *merge* del código, es decir, podemos mezclar las distintas partes de un mismo archivo facilitando la integración de los cambios realizados por distintas personas.

Lo que se hace con los repositorios es tomar «fotografías» del estado actual de los archivos, después podemos comparar distintas fotografías para ver las diferencias entre ellas. A estas fotografía la llamamos «commit», dichos *commits* suelen ser enviados a un servidor remoto para el tema de la colaboración y para mantener copias a prueba de fallos en una computadora local (en esta guía utilizaremos github); para hacer esto se realiza el *push* de las fotografías hacía el servidor. Las personas que estén «conectadas» a ese repositorio (realmente solo realizan copias o un *clone* del mismo) pueden obtener los nuevos *commits* haciendo *pull* desde el servidor.

Otro aspecto a mencionar es la existencia de las ramas o *branch* dentro de un repositorio, estas *branches* son versiones paralelas del mismo repositorio, las cuales podemos visualizar como ramificaciones del repositorio original en un momento en especifico y que podemos, si queremos, volver a integrar en un futuro. Estas ramas suelen utilizarse para diferenciar versiones y para llevar el control de características que se quieran añadir al proyecto (para hacer pruebas sin poner en riesgo al código principal, generalmente en la rama *master*).

Comandos

Ahora que sabemos la teoría de como funciona un repositorio, veremos como crear y manejar uno. Para ello es necesario que nos posicionemos en la carpeta que queremos controla e indicar a *git* que queremos monitorearla.

```
[moy@AfroPC ~/repo]$ git init
Initialized empty Git repository in /home/moy/repo/.git/
```

Nota: El repositorio no tiene por que estar vacío

No solamente podemos crear repositorios desde cero, con frecuencia nos interesará trabajar sobre repositorios ya creados por la comunidad o por algún colega.

```
[moy@AfroPC ~/repo]$ git clone https://github.com/humanytek-team/doc
Cloning into 'doc'...
remote: Enumerating objects: 31, done.
```

(continué en la próxima página)

(proviene de la página anterior)

```
remote: Counting objects: 100% (31/31), done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 31 (delta 6), reused 26 (delta 3), pack-reused 0
Unpacking objects: 100% (31/31), done.
```

Si se trata de un repositorio del que no tenemos permisos de escritura (como los proporcionados por la comunidad), se puede utilizar la URL del mismo (*HTTPS*); sin embargo, se recomienda que si se desea hacer el *clone* de un repositorio al cual haremos *push* se haga mediante el uso de su identificador de *SSH*, esto con la finalidad de hacer más segura la transferencia y evitar tener que colocar las credenciales de la plataforma cada vez que queramos hacer alguna transacción. Para todo esto es necesario haber configurado inicialmente nuestro cliente de *git* con nuestro nombre, correo y, en caso de ser necesario, nuestras llaves *SSH*.

Nota: Se recomienda que los repositorios dentro de un servidor no sean clonados mediante SSH, al menos no sin una contraseña para autorizar las transacciones.

Una vez generado o clonado el repositorio

Podemos llevar el rastreo del estado de nuestros archivos, para ello se utiliza el comando *git status*, el cual nos indicará que archivos han sufrido modificaciones desde el ultimo commit del que se tenga registro.

Para añadir un archivo a nuestra fotografía (*commit*) usamos el comando *git add FILE*, donde *FILE* es el archivo (o archivos) que queremos añadir (también podemos usar . para añadir todo). Una vez añadidos a la escena (*stage*) podemos realizar la fotografía con *git commit -m "MESSAGE"*, la parte de *-m "MESSAGE"* (donde *MESSAGE* es el mensaje que queramos) es opcional, de no colocarla se nos abrirá un editor de texto en el cual debemos introducir la descripción del commit (se recomienda que sea breve y se suelen utilizar etiquetas como *[IMP]*, *[ADD]*, *[REM]*, *[REF]*, etc para indicar rápidamente la naturaleza del mismo).

Después de realizar varios *commits* (puede ser solo uno) podemos mandar nuestros *commits* al servidor con *git push*, si realizamos la conexión mediante *HTTPS* nos solicitará las credenciales de la plataforma, si fue con *SSH* solo nos pedirá la contraseña para hacer uso de la llave (en caso de que lo hayamos indicado).

Si queremos revertir los cambios realizados en un archivo (antes de hacer el *commit*) podemos hacer uso del comando *git checkout FILE*, con esto regresaremos el archivo a como estaba en el ultimo *commit*. Este comando también nos sirve para la creación y manejo de las ramas con la siguiente estructura: *git checkout -b BRANCH*, esto nos moverá (o creará) a la rama que indiquemos.

Apéndices

- https://help.github.com/en/articles/connecting-to-github-with-ssh

1.4.3 PostgreSQL

Duración: 30 min.

Durante este curso se verán temas relacionados a PostgreSQL y SQL en general. Cabe mencionar que es muy rara la ocasión en la que se requiere acceder al gestor de PostgreSQL dentro del mundo del desarrollo de módulos de Odoo.

Requisitos previos

- Cliente PostgreSQL 9.4 o superior

Temario

Como empezar

Lo primero que debemos hacer para trabajar con las BD de PostgreSQL es entrar a la interfaz del mismo, para ello utilizamos el siguiente comando.

```
[moy@AfroPC ~]$ psql -d DATABASE
psql (9.6.11)
Type "help" for help.

grqz=#
```

Nota: Si no tenemos una BD, podemos crearla con *createdb DATABSE*.

Tablas

Dentro de la BD se encuentran las tablas de nuestra aplicación (más adelante veremos que suelen representar a un objeto con todas sus propiedades y, en ocasiones, las relaciones que guardan con otros objetos). Para listar las tablas de una BD dentro de PostgreSQL usamos el comando *dt*, esto nos listara las tablas de dicha BD.

Interactuar con la BD

Una vez que estemos dentro de la BD hay varías cosas que podemos hacer con los registros dentro de las tablas, las mas comunes son realizar consultas (*SELECT*), actualizar registros existentes (*UPDATE*), añadir nuevos registros (*INSERT*) y eliminar los mismos (*DELETE*).

Para cerrar la interfaz podemos utilizar el comando *q* o la combinación de teclas *Ctrl-D*

Apéndices

- http://www.postgresqltutorial.com/

1.4.4 Python

Duración: 180 min.

Este curso se lleva a cabo siguiendo gran parte de la guía oficial de Python y su propósito es aprender lo necesario para el desarrollo de módulos en el mundo de Odoo, por lo que no se tocan temas relacionados a la entrada y salida de datos mediante consola o el uso de archivos u otros artefactos ajenos.

Requisitos previos

- Python 3.5 o superior
- Editor de textos

Temario

Apéndices

- https://docs.python.org/3.7/tutorial/