Capacitación Git
Módulo 5

Repositorios distribuidos

por Gonzalo Barro Gil, fuente https://git-scm.com/book/es/v2

https://github.com/GonzaloBarroGil/git-course

Protocolos de transferencia

- Vimos previamente que git registra referencias remotas en forma de path o url.
- Existen cuatro protocolos para transferencia de datos entre repositorios Git:
  - Local
    - FS o NFS. Acceso por path o por url file://
  - HTTP/S
    - Usuario y contraseña o acceso público. Acceso por url
  - SSH
    - Criptografía asimétrica, Acceso seguro por ssh
  - Git
    - Alta velocidad sin autenticación. Acceso por git-daemon

Remotes

- Vimos en el módulo anterior que eran los branches remotos y que se obtenía una referencia a los mismos al clonar un repositorio.

- Un **remote** es una referencia con nombre. En un entorno distribuido puede haber más de una, por ejemplo para los clones de otros colaboradores:

```
git remote -v
origin     git@github.com:GonzaloBarroGil/git-course.git (fetch)
origin     git@github.com:GonzaloBarroGil/git-course.git (push)
```

  - En este caso se define la misma url tanto para envío como para recepción de datos con protocolo ssh (git@server:repo.git).

- Cuando se clona un proyecto se crea una por defecto llamada **origin**. Se pueden definir más con otro nombre con:

```
git remote add clonJuancho https://github.com/Juancho/git-course.git
```

  - De esta manera se define el remote clonJuancho y el protocolo HTTPS para lo cual (al menos para push) requerirá usuario (habilitado por Juancho) y contraseña.

```
git remote -v
origin     git@github.com:GonzaloBarroGil/git-course.git (fetch)
origin     git@github.com:GonzaloBarroGil/git-course.git (push)
clonJuancho https://github.com/Juancho/git-course.git (fetch)
clonJuancho https://github.com/Juancho/git-course.git (push)
```

- Los remotes pueden modificarse con :

```
git remote set-url clonJuancho git@github.com:nuevoJuancho/git-course.git
```

Local remotes

- Se puede definir una referencia con el protocolo Local hacia una carpeta del mismo FS o de un NFS, en este caso la seguridad viene dada por la administración del propio sistema.

- **Ejercitación**:
  - Seguir la guía descripta en https://github.com/GonzaloBarroGil/git-course/blob/master/LocalRemotes.rst

Protocolo SSH

- SSH es un protocolo de comunicación que establece un canal seguro entre un cliente y un servidor. De esta manera el cliente puede tomar control del servidor

- Utiliza varios métodos, pero en el caso de Git se utiliza criptografía asimétrica en la cual se utilizan dos claves:
  - Una pública para cifrar mensajes
  - Una privada para descifrarlos

- En el caso de git se envía la clave pública al servidor para que la autorice y habilite el inicio de sesión del cliente.

- Utilización de SSH:
  - Linux: Suele venir instalado en la mayoría de las distribuciones y se utiliza en el terminal.
  - Windows: No viene instalado por defecto y se debe utilizar un cliente como Putty.

Instalación SSH

- Linux: Suele venir instalado. Se puede verificar con:

```
ssh -V
```

  - Si no muestra una versión o da error, el paquete se llama openssh-client. En distribuciones Debian:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install openssh-client
```

- Windows: No viene instalado. Se puede instalar desde PowerShell (como administrador) pero antes hay que verificar si está disponible:

  - ```
    Get-WindowsCapability -Online | ? Name -like 'OpenSSH*'
    ```
    ```
    Name  : OpenSSH.Client~~~~0.0.1.0
    State : NotPresent
    Name  : OpenSSH.Server~~~~0.0.1.0
    State : NotPresent
    ```

  - Buscar la versión correcta del cliente e instalarlo:

  - ```
    Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0
    ```

- En versiones anteriores de Windows se debe instalar un cliente tipo Linux, como Putty:

  https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

  - Ir a la sección Package Files y seleccionar el MSI ('Windows Installer') para la arquitectura correspondiente (64-bit o 32-bit).

Generar par de claves

- Verificar claves existentes:
  - `ls -a ~/.ssh`
    Mostrará un par de archivos como id_rsa y id_rsa.pub, o id_ed25519 y ed25519.pub
- Generar nuevo par de claves:
  - `ssh-keygen -t rsa -b 4096 -C "email@ejemplo.com"`
    Esto generará un par de claves con el método RSA.
  - `ssh-keygen -t ed25519 -C "email@ejemplo.com"`
    Esto generará un par de claves con el método ed25519.
- El proceso nos solicitará una ruta y un nombre de archivo para la clave. Es recomendable usar el valor por defecto (~/.ssh/id_rsa)
- También nos permitirá definir una password. Para no tener que ingresar la password cada vez que se utilice SSH se puede agregar la clave al agente SSH:
  - `eval "$(ssh-agent -s)"`
  - `ssh-add ~/.ssh/id_rsa`

Compartir clave

- Recordemos que entre nodos sólo se comparte la clave pública, nunca la privada.
- La clave pública suele terminar con .pub en el nombre de archivo.
- Se debe compartir todo el contenido del archivo, por ejemplo para un clave rsa llamada id_rsa:
  - `cat ~/.ssh/id_rsa.pub`

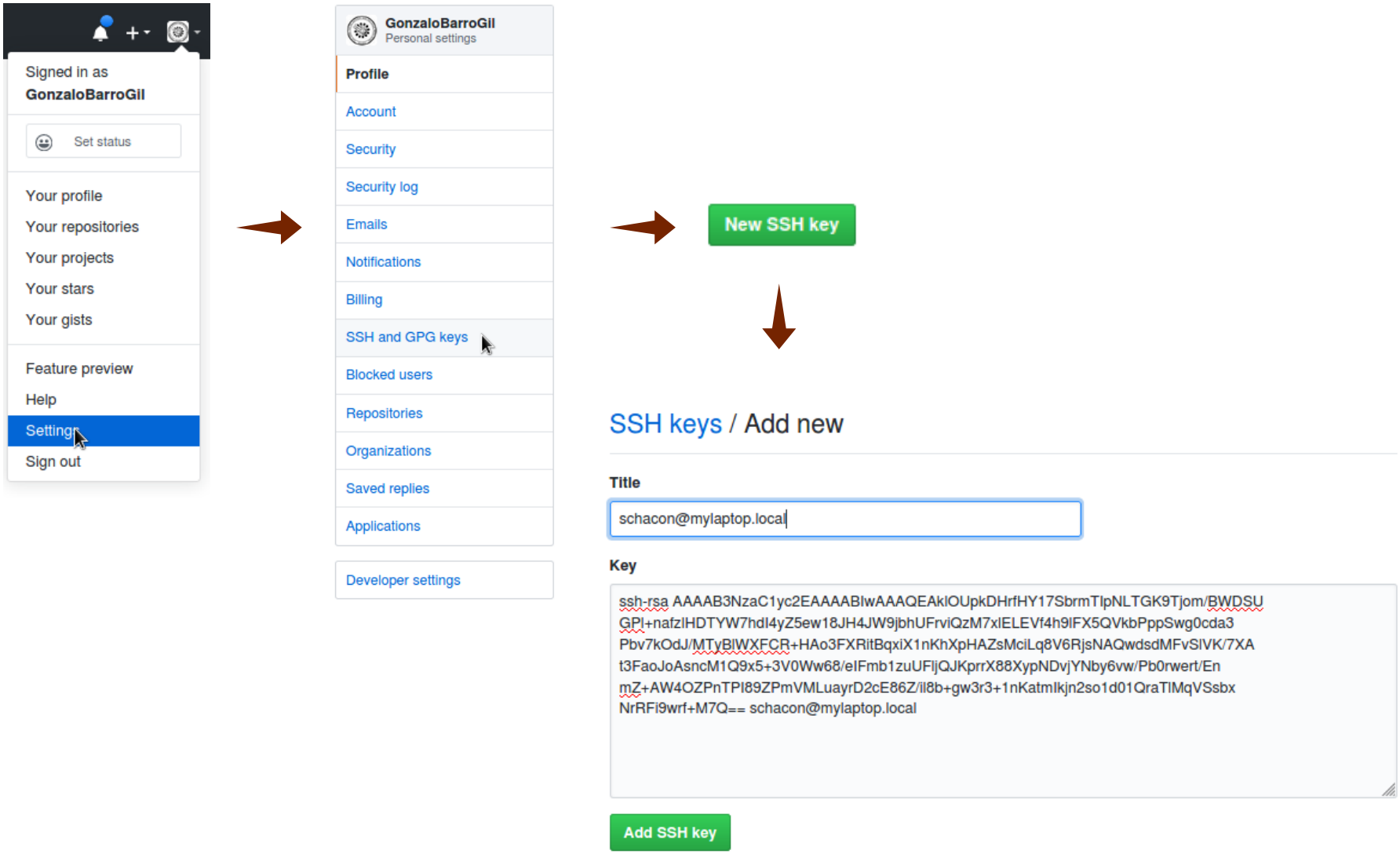
    Mostrará el contenido de la clave, por ejemplo:

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAklOUpkDHrfHY17SbrmTIpNLTGK9Tjom/BWDSU
GPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7xlELEVf4h9lFX5QVkbPppSwg0cda3
Pbv7kOdJ/MTyBlWXFCR+HAo3FXRitBqxiX1nKhXpHAZsMciLq8V6RjsNAQwdsdMFvSlVK/7XA
t3FaoJoAsncM1Q9x5+3V0Ww68/eIFmb1zuUFljQJKprrX88XypNDvjYNby6vw/Pb0rwert/En
mZ+AW4OZPnTPI89ZPmVMLuayrD2cE86Z/il8b+gw3r3+1nKatmIkjn2so1d01QraTlMqVSsbx
NrRFi9wrf+M7Q== schacon@mylaptop.local
```

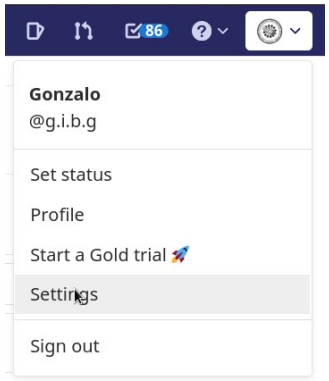
    Se debe copiar desde ssh-rsa … hasta … schacon@mylaptop.local, inclusive.

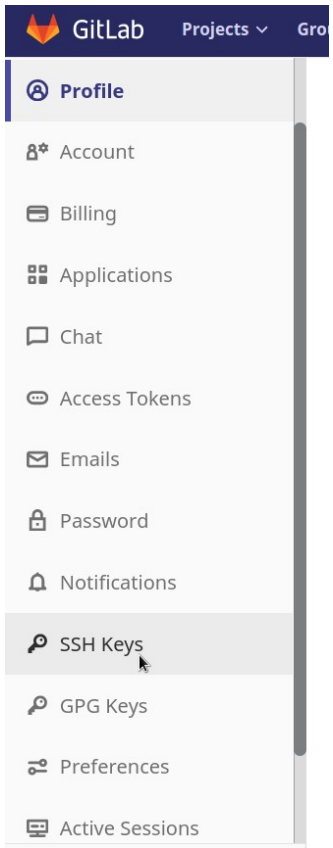
- Luego se registra en el servidor, normalmente logueado previamente con usuario y contraseña .

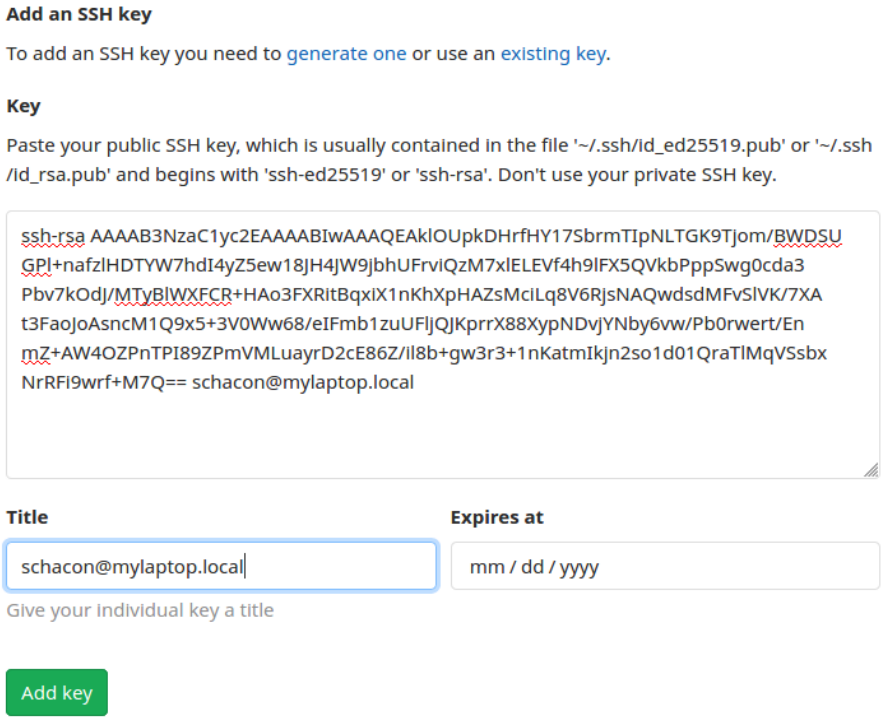
Registrar clave en GitHub



Registrar clave en GitLab







Protocolo HTTPS

- Acceso público
  - Clonado
  - Descarga
- Usuario autorizado
  - Usuario
  - Contraseña
- Almacenar credenciales
  - Modo **default**:
    - En cada operación de transferencia (clone, push, fetch, pull) Git solicitará usuario y contraseña.
  - Modo **cache**:

    ```
    git config --global credential.helper cache --timeout 900
    ```

    - Las credenciales se guardan en memoria por unos minutos
  - Modo **store**:

    ```
    git config --global credential.helper store --file ~/.my-credentials
    ```

    - Las credenciales se guardan en un archivo en disco
- Manejadores de credenciales propietarios:
  - Mac:
    - git-credential-osxkeychain
  - Windows:
    - https://github.com/Microsoft/Git-Credential-Manager-for-Windows