relevant.

Git & GitHub

git

Índice Sesión 7

releevant.

1. **Comandos Avanzados**
   - Stash
   - Rebase
2. **Fichero .gitignore**
3. **Protocolos**
4. **Fork**
5. **GitFlow**

Stash

relevant.

1. Desde master crear una nueva rama “fix2022”
2. Listar las ramas
3. En master hacer algún cambio (crear fichero o modificar fichero) que contenta el texto “incompleto”
4. Cambiar a la nueva rama “fix2022”

¿Cómo evitaríamos perder los cambios?

Stash

relevant.

Save unstaged changes

```
git stash
```

List all saved changes

```
git stash list
```

Apply one change

```
git stash pop [STASH]
```

Stash

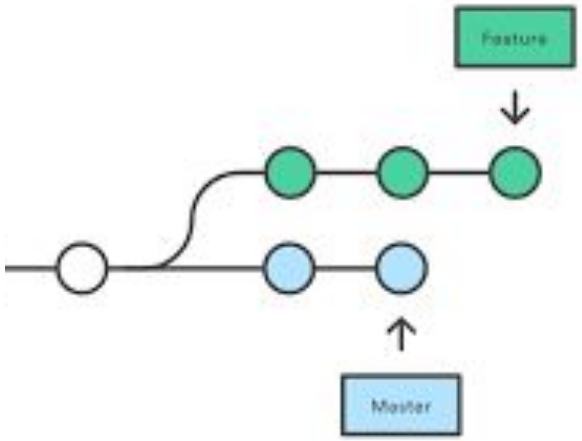
relevant.

1. Salvar los cambios en la rama master
2. Cambiar a la nueva rama “fix2022”
3. En “fix2022” hacer algún cambio (crear fichero o modificar fichero) que contenta el texto “arreglo”
4. Subir los cambios a repositorio remoto
5. Cambiar a la rama master
6. Recuperar los cambios guardados con anterioridad

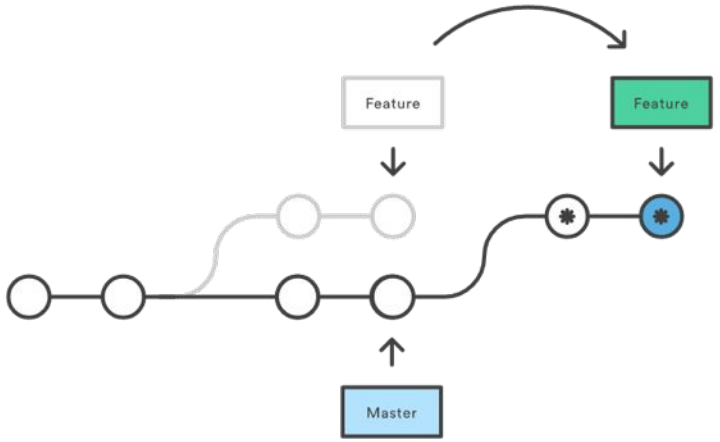
releevant.

Cambiar la base de una rama

- Una rama se usa para hacer desarrollos separados de la rama master
  - Al finalizar el desarrollo de la rama, se puede integrar con master u otra rama
- Cambiar la base de una rama permite también integrar desarrollos
  - Cambiar de base (**rebase**) integra los desarrollos linealmente (muy limpio)
    - Pero **elimina la historia** de ramas utilizadas e integradas para el desarrollo



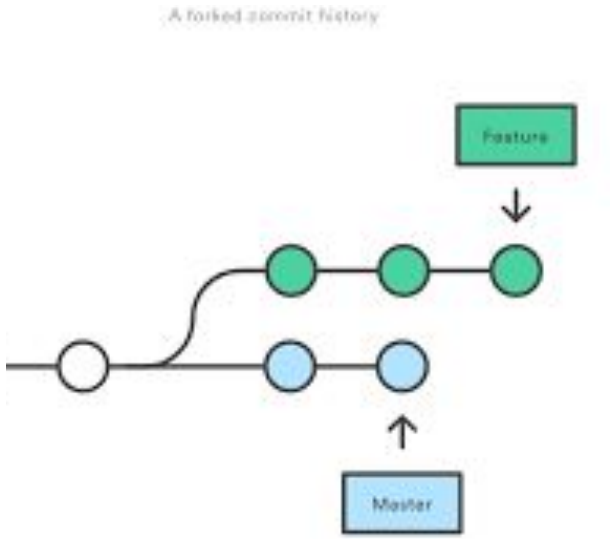
After Rebasing Onto Master

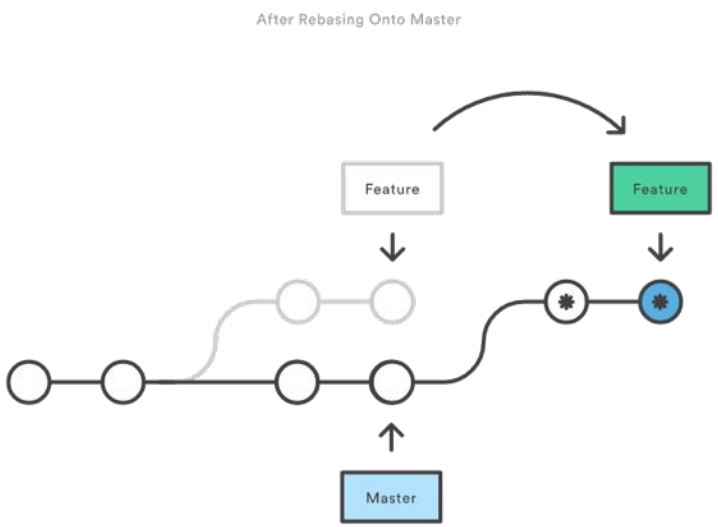


Rebase

Estando en una rama diferente a master (mirama):

- git checkout master
- Añadir fichero
- Modificar fichero
- git checkout mirama
- git rebase master





Rebase

.gitignore

Raw

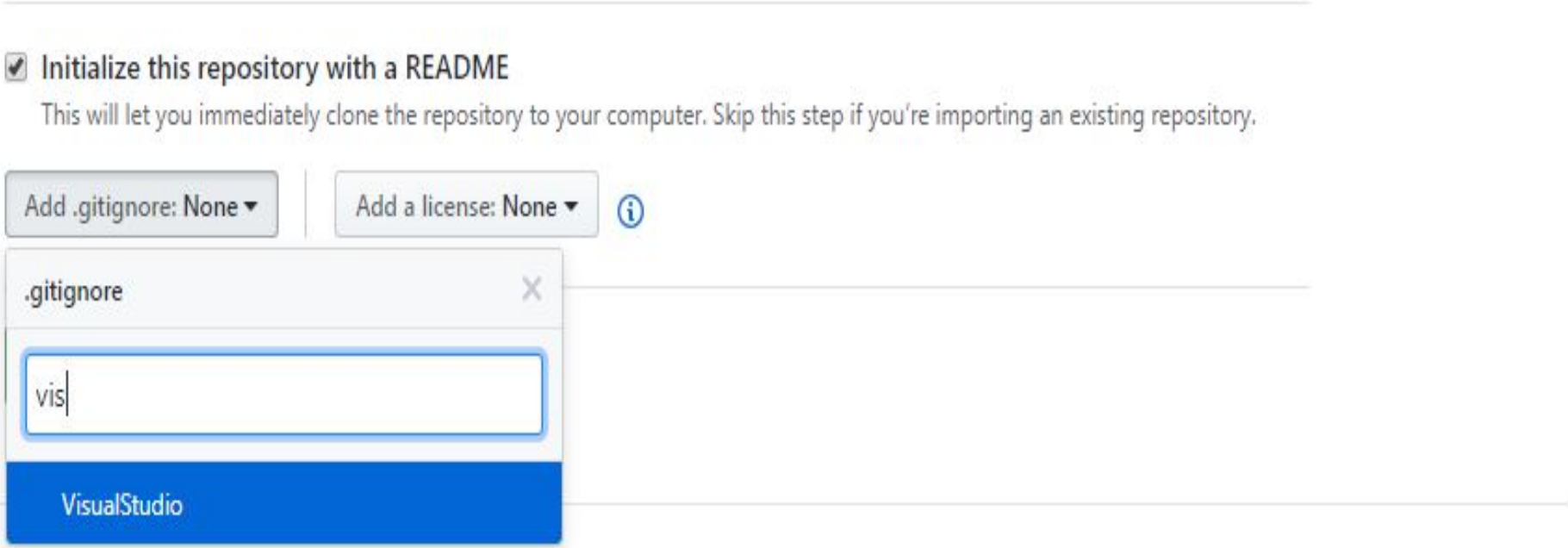
releevant.

```
# Compiled source #
###################
*.com
*.class
*.dll
*.exe
*.o
*.so

# Packages #
############
# it's better to unpack these files and commit the raw source
# git has its own built in compression methods
*.7z
*.dmg
*.gz
*.iso
*.jar
*.rar
*.tar
*.zip
```

gitignore

releevant.



gitignore

releevant.

```
# ignore all logs
*.log
```

You can use \ to escape .gitignore pattern characters if you have files or directories containing them:

```
# ignore the file literally named foo[01].txt
foo\[01\].txt
```

gitignore

gitignore

releevant.

https://es.atlassian.com/git/tutorials/saving-changes/gitignore#git-ignore-patterns
https://gist.github.com/octocat/9257657

Protocolos

Protocolo Local

El más básico es el *Protocolo Local*, donde el repositorio remoto es simplemente otra carpeta en el disco. Se utiliza habitualmente cuando todos los miembros del equipo tienen acceso a un mismo sistema de archivos, como por ejemplo un punto de montaje NFS, o en los casos en que todos se conectan al mismo ordenador. Aunque este último caso no es precisamente el ideal, ya que todas las instancias del repositorio estarían en la misma máquina; aumentando las posibilidades de una pérdida catastrófica.

Si dispones de un sistema de archivos compartido, podrás clonar (clone), enviar (push) y recibir (pull) a/desde repositorios locales basado en archivos. Para clonar un repositorio como estos, o para añadirlo como remoto a un proyecto ya existente, usa el camino (path) del repositorio como su URL. Por ejemplo, para clonar un repositorio local, puedes usar algo como:

```
$ git clone /opt/git/project.git
```

O como:

```
$ git clone file:///opt/git/project.git
```

releevant.

Protocolos

El Procotolo SSH

Probablemente, SSH sea el protocolo más habitual para Git. Debido a disponibilidad en la mayor parte de los servidores; (pero, si no lo estuviera disponible, además es sencillo habilitarlo). Por otro lado, SSH es el único protocolo de red con el que puedes facilmente tanto leer como escribir. Los otros dos protocolos de red (HTTP y Git) suelen ser normalmente protocolos de solo-lectura; de tal forma que, aunque los tengas disponibles para el público en general, sigues necesitando SSH para tu propio uso en escritura. Otra ventaja de SSH es el su mecanismo de autentificación, sencillo de habilitar y de usar.

Para clonar un repositorio a través de SSH, puedes indicar una URL ssh:// tal como:

```
$ git clone ssh://user@server/project.git
```

Protocolos

releevant.

El Protocolo Git

El protocolo Git es un demonio (daemon) especial, que viene incorporado con Git. Escucha por un puerto dedicado (9418), y nos da un servicio similar al del protocolo SSH; pero sin ningún tipo de autentificación. Para que un repositorio pueda exponerse a través del protocolo Git, tienes que crear en él un archivo 'git-daemon-export-ok'; sin este archivo, el demonio no hará disponible el repositorio. Pero, aparte de esto, no hay ninguna otra medida de seguridad. O el repositorio está disponible para que cualquiera lo pueda clonar, o no lo está. Lo cual significa que, normalmente, no se podrá enviar (push) a través de este protocolo. Aunque realmente si que puedes habilitar el envio, si lo haces, dada la total falta de ningún mecanismo de autentificación, cualquiera que encuentre la URL a tu proyecto en Internet, podrá enviar (push) contenidos a él. Ni que decir tiene que esto solo lo necesitarás en contadas ocasiones.

Protocolos

releevant.

El protocolo HTTP/S

Por último, tenemos el protocolo HTTP. Cuya belleza radica en la simplicidad para habilitarlo. Basta con situar el repositorio Git bajo la raiz de los documentos HTTP y preparar el enganche (hook) 'post-update' adecuado. (Ver el Capítulo 7 para detalles sobre los enganches Git.) A partir de ese momento, cualquiera con acceso al servidor web podrá clonar tu repositorio. Para permitir acceso a tu repositorio a través de HTTP, puedes hacer algo como esto:

```
$ cd /var/www/htdocs/
$ git clone --bare /path/to/git_project gitproject.git
$ cd gitproject.git
$ mv hooks/post-update.sample hooks/post-update
$ chmod a+x hooks/post-update
```

Y eso es todo. El enganche 'post-update' que viene de serie con Git se encarga de lanzar el comando adecuado ('git update-server-info') para hacer funcionar la recuperación (fetching) y el clonado (cloning) vía HTTP. Este comando se lanza automáticamente cuando envias (push) a este repositorio vía SSh; de tal forma que otras personas puedan clonarlo usando un comando tal que:

```
$ git clone http://example.com/gitproject.git
```

About forks

A fork is a copy of a repository that you manage. Forks let you make changes to a project without affecting the original repository. You can fetch updates from or submit changes to the original repository with pull requests.

Any user or organization on GitHub can fork a repository. Forking a repository is similar to copying another repository, with two major differences:

- You can use a pull request to suggest changes from your fork to the original repository, also known as the *upstream* repository.
- You can bring changes from the upstream repository to your local fork by synchronizing your fork with the upstream repository.

Deleting a fork does not delete the original upstream repository. In fact, you can make any changes you want to your fork--add collaborators, rename files, generate GitHub Pages--with no effect on the original.

releevant.

Fork

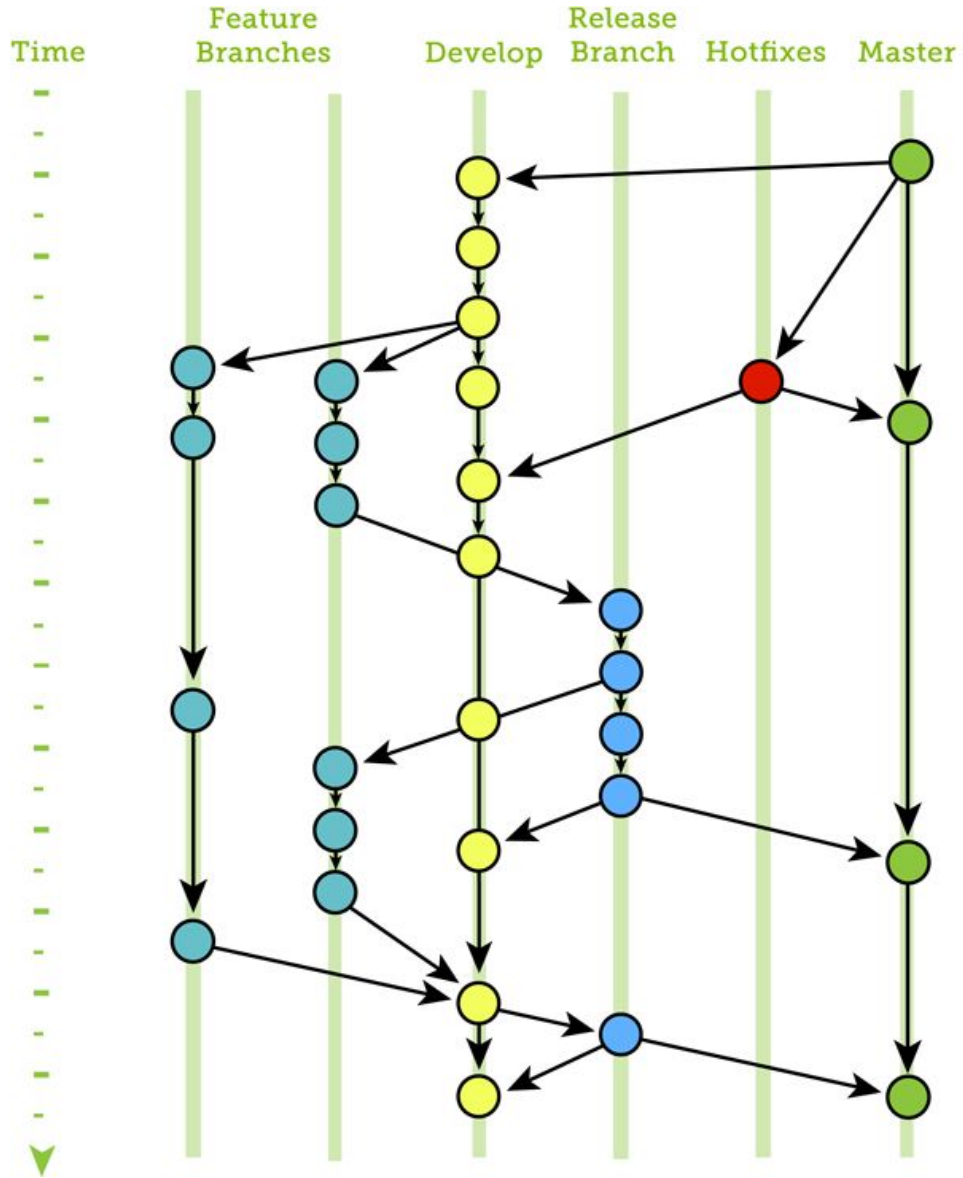
Copia un repositorio albergado en GitHub a otra cuenta (para contribuir).

* En GitHub, navegad al repositorio de algún compañero.

*En la esquina derecha de arriba, clicar en Fork.

*Crear un fichero nuevo y subirlo al repo del compalero y solicitar Pull-Request.

*El compañero acepta y su Código se actualiza.



releevant.

GitFlow

releevant.

Referencias

- https://git-scm.com/book/es/v2/
- https://help.github.com/articles/getting-started-with-writing-and-formatting-on-github/
- https://blog.axosoft.com/merge-conflict-tool/


KEEP
CALM
AND
CARRY ON
CODING