

# Trabalho Prático 2

## Comunicação por Computadores

### PL6 Grupo 66

Inês Vicente<sup>[a93269]</sup>, Jorge Melo<sup>[a93308]</sup>, and Mariana Rodrigues<sup>[a93229]</sup>

Universidade do Minho

**Keywords:** UDP · TCP · HTTP · Controlo de Fluxo · Controlo de Congestão.

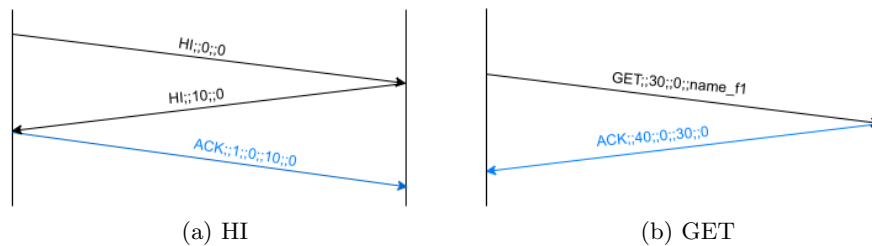
## 1 Introdução

O objetivo deste TP é implementar um sistema de sincronização de pastas sobre uma conexão UDP. A principal vantagem do UDP é ser extremamente rápido, mas isso faz com que não haja controlo de perdas. Sendo assim, somos nós que temos de fazer esse controlo.

Para além disso, também somos nós que definimos o formato dos pacotes a enviar, definindo, por exemplo, diferentes tipos de dados.

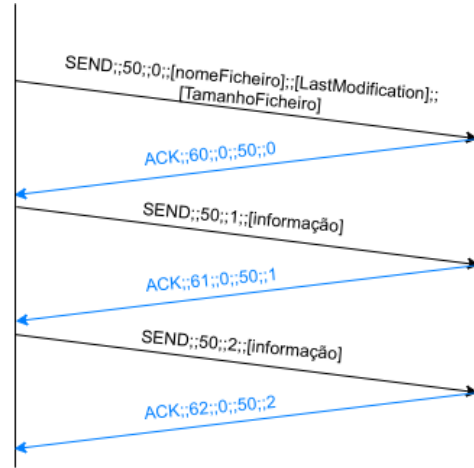
Neste trabalho, tentámos aplicar os diferentes conhecimentos que aprendemos ao longo das aulas teóricas.

## 2 Arquitetura da solução

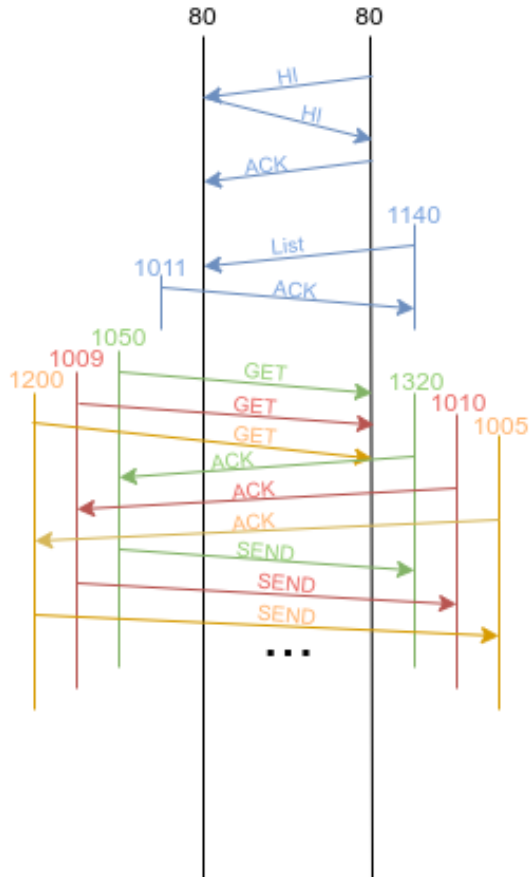




(c) List



(d) SEND



(e) Nossa implementação

### 3 Especificação do protocolo

#### 3.1 Início da ligação

Para o início da ligação, é aberto um *socket*, seguido do envio do pacote **HI**. Cada um dos *peers* tem acesso à mesma lista de "segredos partilhados", necessários para autenticação. O primeiro *peer* a enviar um **HI** vai tirar um elemento aleatório dessa lista e anexá-lo ao **HI**. O outro *peer* vai a essa lista procurar o elemento correspondente ao elemento recebido e anexa-o ao **HI** que vai enviar de volta. Quando voltamos ao primeiro *peer*, este verifica se o elemento recebido corresponde ao elemento correspondente enviado. Se corresponder, envia um **ACK** e a conexão fica estabelecida. Senão, ele simplesmente cancela a conexão.

#### 3.2 Fim da ligação

O término de ligação é estabelecido através do envio do pacote **Bye** e, só posteriormente, fecha-se o *socket*.

Este término de ligação pode ser forçado, isto é, pode ocorrer sempre que o *socket* em questão não tenha recebido um suposto pacote num dado tempo de tolerância (*Round Trip Time*). No caso do destinatário não ter recebido o pacote **Bye**, este, por defeito, irá terminar a sua ligação quando o seu tempo de tolerância findar.

#### 3.3 Formato das mensagens protocolares

Neste protocolo, por defeito, o tamanho máximo do pacote é de 1460 *bytes*. Este protocolo suporta 6 tipos de pacotes: **Hi**, **Bye**, **Ack**, **List**, **Get**, **Send**.

Todos os pacotes de dados enviados deverão respeitar o seguinte formato:



Fig. 1: Pacote de Dados

O primeiro campo caracteriza-se pela identificação do tipo de pacote que é.

Este encontra-se sub-dividido, sendo o primeiro *bit* (o extremo esquerdo) reservado para conseguirmos ter uma *flag*.

Esta *flag*, servirá para conseguirmos identificar se o pacote é o último/único (*bit* a 1), ou se, num dado contexto, ainda vamos receber mais pacotes do mesmo género (*bit* a 0).

O número de sequência do pedido é atribuído por cada transmissor da mensagem, garantindo, com isto, ser possível identificar a que pacote corresponde o pedido.

O campo relativo ao número de sequência de segmento serve para, no caso de o pedido ter que ser segmentado, isto é, ser enviado em vários pacotes, conseguirmos garantir a ordem dos pacotes recebidos dum dado pedido. Com isto, garantimos a ordem dos pedidos enviados e que nenhum pacote é perdido.

O último campo é particular a cada um dos pacotes.

### 3.3.1 Hi

Pacote associado ao início de conexão. Na parte de dados, ele terá uma *string* que será usada para a autenticação, como explicado anteriormente no **Início da ligação**.

### 3.3.2 Bye

Pacote associado ao término de conexão. Neste caso, o campo de dados não conterá nenhuma informação.

### 3.3.3 Ack

Pacote associado à confirmação de receção de um pacote recebido previamente.

Nos dados particulares desta mensagem, encontra-se presente o *byte* de sequência do pedido e o *byte* de sequência do segmento do pacote que queremos confirmar que já foi recebido.

### 3.3.4 List

Pacote associado ao envio dos nomes e data de edição dos ficheiros de uma das pastas para a outra. No campo de dados, ele terá uma *string* no formato *nomeFile1;;dataFile1;;nomeFile2;;dataFile2;;....* Estas informações serão usadas posteriormente para comparar os ficheiros entre as duas pastas. Verificando, assim, se há ficheiros que estejam apenas numa das pastas, através do nome. Se o ficheiro existir em ambas as pastas, teremos que comparar a data de última edição para ver se é necessário atualizar esse mesmo ficheiro.

### 3.3.5 Get

Pacote associado ao pedido de um determinado ficheiro.

Nos dados particulares desta mensagem encontra-se presente o nome do ficheiro que se pretende receber.

### 3.3.6 Send

Pacote associado ao envio de um ficheiro.

No primeiro pacote de um pedido **Send** encontra-se nos dados particulares uma *string* no seguinte formato: *nomeFicheiro;;LastModification;;TamanhoFicheiro*. Nos restantes pacotes desse pedido **Send** teremos nos dados particulares os *bytes* do ficheiro em si.

### 3.4 Controlo de Entrega

Para conseguirmos garantir que nenhum pacote é perdido, foi decidido que cada tipo de pedido iria ter um número de sequência, não podendo existir dois ou mais pedidos a serem enviados com o mesmo número de sequência. Fora isso, para conseguirmos garantir a ordem dos dados enviados, tal como no **TCP**, numeramos os pacotes de dados através do número de sequência de segmento.

Sempre que é enviado um pacote de dados de um dado pedido, terá de ser guardado o seu número de segmento. Sendo assim, o próximo pacote a ser enviado terá que ter o mesmo número de pedido que o anterior (pacotes do mesmo pedido) e o seu número de segmento será igual ao anterior guardado +1.

No caso do emissor, para este ter conhecimento que o outro lado da ligação recebeu os pacotes enviados por este, deve guardar o número de sequência e de segmento do pacote enviado. Com isto, ele espera, de seguida, receber um **ACK** com esse número de sequência e com o número de segmento.

### 3.5 Pacotes Duplicados

Na ocorrência de serem recebidos várias réplicas do mesmo pacote, todos serão descartados com exceção do primeiro.

### 3.6 Controlo de Fluxo

Este protocolo é baseado em *stop-and-wait*, um método *feedback-based* de controlo de fluxo. Isto significa que o *sender* só envia outro pacote quando recebe um sinal do *receiver*, neste caso, um **ACK**. Isso vai garantir que o *receiver* nunca está sobrecarregado.

### 3.7 Controlo de Congestão

No que toca ao controlo de congestão, decidimos que cada pacote terá 1460 *Bytes* de tamanho. Apesar de haver, possivelmente, várias *threads* a mandar pacotes, cada *thread* manda os pacotes sequencialmente, sendo pouco provável haver uma grande congestão na rede.

### 3.8 Controlo de Perdas

Quando o emissor não recebe o **ACK** que esperava (o **ACK** com número de sequência e de segmento do pacote enviado), este volta a enviar o pacote. Se isto acontecer 3 vezes consecutivas, é fechada a conexão.

## 4 Implementações

O nosso programa terá duas grandes funcionalidades. Para isso, mal o programa é iniciado, este irá criar 2 *threads*. Cada qual ficará responsável por estas funcionalidades, respetivamente:

- **Communication** : responsável por realizar a sincronização entre duas pastas de *peers* diferentes;
- **Listening** : conseguir responder a pedidos **HTTP**, apresentando o estado de funcionamento atual.

Como em todo o trabalho foram usadas *threads*, foi crucial existirem 2 classes partilhadas por todas estas:

- **Information**  
Responsável por conter a informação geral de todo o programa, como por exemplo, quantos ficheiros já foram enviados, recebidos, um *boolean* que nos informará se o programa já terminou, entre outros.
- **Log**  
Responsável por escrever e atualizar os ficheiros de *log*.  
Classe que irá ser falada mais em detalhe *a posteriori*.

### 4.1 Listening

Como referido anteriormente, esta *thread* é a responsável por responder a pedidos **HTTP**.

Enquanto que o *boolean* que nos informa se o programa terminou estiver a *false*, a *thread* ficará à espera de receber um qualquer pedido através da porta 80. Recebendo um pedido, irá passar ao seu tratamento.

O nosso programa foi construído para saber responder a estes 2 tipos de pedidos: **GET** e **SEND**.

Este só conhece o:

- **/** : Apresenta a pasta que esta a ser sincronizada com alguma informação adicional.
- **/log** : Apresenta os ficheiros dos **log** e mais alguma informação geral acerca do funcionamento do programa.

Qualquer outro tipo de pedido irá responder com um *error 501 Not Implemented*.

### 4.2 Communication

Esta *thread* é a responsável por toda a sincronização, isto é, pelo processamento de pedidos, atualizar os ficheiros de *logs*, e um pequeno menu que adicionará algumas funcionalidades extras à nossa sincronização.

Para tal, é de destacar outras 4 *threads*.

- **Log**  
Responsável por atualizar todos os ficheiros *logs* consoante os pacotes que vai recebendo e enviando.
- **Menu**  
Responsável por apresentar um menu com algumas funcionalidades extras.
- **SynchronizedDirectory**  
Responsável por manter a pasta a sincronizar atualizada, esta apenas envia mensagens do tipo **List**.
- **ReceivedAndTreat**  
Aqui estará a maior parte da complexidade do trabalho. Esta *thread* encontra-se responsável por receber diferentes tipos de pedidos, tratando deles ao mesmo tempo.

```
ReceivedAndTreat receivedAndTreat = new
    ↳ ReceivedAndTreat(status, socket, pathDir,
    ↳ clientIP, seqPedido, log);
SynchronizeDirectory synchronizeDirectory =
    new SynchronizeDirectory(status, socket,
        ↳ pathDir, clientIP, port, seqPedido, log);
RunMenu menu = new RunMenu(status, port, clientIP,
    ↳ log, seqPedido);
Thread[] threads = new Thread[4];
threads[0] = new Thread(receivedAndTreat);
threads[1] = new Thread(synchronizeDirectory);
threads[2] = new Thread(log);
threads[3] = new Thread(menu);
```

#### 4.2.1 Menu

Tal como denotado anteriormente, esta *thread* é responsável por apresentar um menu com algumas opções, entre elas:

- **adicionar nome de ficheiro para ignorar**: caso não se queira sincronizar um ficheiro específico, podemos usar esta opção para o meter numa lista de ficheiros a ignorar;
- **adicionar nome de ficheiro para sincronizar**: tendo ficheiros na lista de ficheiros a ignorar, é possível voltar a pô-los a sincronizar novamente;
- **Ver ficheiros que não estão a ser sincronizados**: mostra a lista de ficheiros a ignorar;
- **Terminar programa**: termina o programa.

De notar que este menu só aparece quando é terminada a primeira sincronização entre as duas pastas.

```
<idor1.conf# cc_bash 10.3.3.2 /home/core/tp2-folder2
Pasta sincronizada deste lado

MENU
0: Adicionar nome de ficheiro para ignorar
1: Adicionar nome de ficheiro para sincronizar
2: Ver ficheiros que não estão a ser sincronizados
3: Terminar programa
```

Fig. 2: Menu

#### 4.2.2 Log

Enquanto há outras *threads* a tratar de enviar e receber ficheiros, esta serve simplesmente para registar em ficheiros as informações dos pacotes a receber e enviar. Esta *thread* foi criada para evitar haver várias *threads* a interagirem todas com o mesmo ficheiro, o que poderia originar problemas.

Existem 3 ficheiros de *logs*:

- **.log**: contém todas as informações dos pacotes enviados;
- **.logReceived**: contém todas as informações dos pacotes recebidos;
- **.logTime**: contém os tempos e débitos dos ficheiros enviados

#### 4.2.3 ReceivedAndTreat

Finalizada a especificação do protocolo, passou-se para a criação de cada um dos tipos de mensagem que se podem ter, todos estes tipos de pedidos implementam a interface **MSG\_interface**. Aqui é de salientar que essa interface obriga a ter estes métodos de destaque:

```
void send() throws IOException, PackageErrorException,
    ↪ AuthenticationFailed, TimeoutMsgException;

void send(DatagramSocket socket) throws IOException,
    ↪ PackageErrorException, TimeoutMsgException;

void received() throws IOException, TimeoutMsgException,
    ↪ PackageErrorException, AckErrorException,
    ↪ AuthenticationFailed;
```

Com isto, foi possível desenvolver a camada responsável pelo transporte totalmente agnóstica ao tipo de mensagem que seria enviada ou recebida.

Sendo assim, todos os pedidos enviados seguiram o formato da **MSG\_interface** e todos os pacotes recebidos serão convertidos para esse mesmo formato. Para isso, todos os pacotes recebidos serão inicialmente verificados, tendo que ter o cabeçalho no formato correto. Essa verificação garante que a mensagem é confiável e de um tipo de pedido conhecido.



Para conseguir responder a vários pedidos em simultâneo, criámos as classes **SendMSWithChangePorts** e **ControlMsgWithChangePorts**. Estas classes tratam de reencaminhar os pedidos recebidos para outras portas e de dividir os pedidos por diferentes *threads*. Operam através da porta 80, e procedem da seguinte forma:

- **ao enviar um pacote**

Cria uma nova *thread*, com um novo *socket*, numa porta diferente, por onde envia o pacote para a porta principal (80) do outro *peer*. Depois, fica à escuta e espera pelo **ACK**. Quando o recebe, muda a porta do cliente para a porta de onde veio o **ACK**, passando a comunicar por aí com esse *peer*, sem ser pela porta principal.

- **ao receber um pacote válido**

É usada uma nova porta que esteja disponível, e envia por essa porta o **ACK**, para a porta de onde foi recebido o pacote. Passa, de seguida, ao processamento do pedido através do *socket* criado outrora.

Com isto, é possível mandar e receber vários pedidos em simultâneo, garantindo que pacotes de pedidos diferentes não se misturem.

#### 4.2.4 SynchronizedDirectory

Para ser possível manter as duas pastas sincronizadas ao longo do tempo, é necessário ir vendo se alguma das pastas sofreu alterações. Para tal, esta *thread* fica exclusivamente responsável por ir mandando pedidos **List**.

Esta, enquanto o *boolean* que informa que o programa terminou se encontrar a *false*, irá, durante um certo espaço de tempo, enviar o pacote **List** para a porta 80 do outro *peer*, através de um novo *socket*. Aí, é enviado um **ACK** e, posteriormente, o pedido será tratado. Ao receber o **ACK**, o *socket* por onde o **List** foi enviado é fechado.

### 4.3 Bibliotecas de Suporte Usadas

```
/** Criação de Pacotes enviados */
import java.net.DatagramPacket;
/** Canal UDP utilizado */
import java.net.DatagramSocket;
/** Resolução de nomes e IP's */
import java.net.InetAddress;

/** Canal TCP utilizado */
import java.net.ServerSocket;

/** Garantir a exclusão mútua entre as diferentes
↪ threads */
```

```
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.locks.Condition;
```

#### 4.4 Resultados e Comparações

Nome do ficheiro	Tamanho (b)	Tempo (ms)	Débito (bits/s)
topo.imn	30211	95	2
rfc7231.txt	235187	218	8
Chapter_3_v8.0.pptx	6140969	1755	27

Nome do ficheiro	Tamanho (b)	Tempo (ms)	Débito (bits/s)
topo.imn	30211	155	1
book.jpg	188533	313	4
rfc7231.txt	235053	256	7
bootstrap-dist.zip	592115	350	13
Chapter_3_v8.0.pptx	6140969	1912	25
wireshark.tar.xz	32335284	7587	34

## 5 Conclusão

Após a conclusão deste trabalho, ficámos a perceber melhor como é feita a comunicação por computadores sobre uma conexão **UDP** e como criar protocolos para diferentes tipo de mensagens. Ao longo do desenvolvimento deste trabalho, fomos tendo dificuldades, especialmente em conseguir mandar e fazer o tratamento de pedidos em simultâneo, sem serem perdidos pacotes.

Dito isto, achamos que conseguimos desenvolver um programa que vai de encontro aos requisitos dos professores, tendo concluído todos os requisitos obrigatórios e ainda alguns opcionais como a sincronização de pastas e subpastas, um menu com opções adicionais de adicionar e remover ficheiros de uma lista de ignorados e um http com funcionalidades extra.

## References

1. **oracle - java** <https://docs.oracle.com/javase/7/docs/api/overview-summary.html>
2. "Computer Networking: A Top-Down Approach", 7th Edition, J. Kurose and K. Ross

## Exemplos de pedidos ao HTTP

**Nota:** Nestes exemplos é sempre mostrada a porta 8080 a ser usado. Posteriormente a porta foi trocada para a 80.

The image is a composite of three screenshots. The top-left screenshot shows two terminal windows. The left terminal is running a menu-driven program in a directory named /home/core/tp2-folder2, with options to add files to ignore or sync, view files, or terminate. The right terminal is running the same program in /home/core/tp2-folder3. The bottom-left screenshot shows a terminal window where a uget command is used to fetch a file from a web server, and a diff command is used to compare the contents of two directories. The right screenshot shows a web browser displaying the status of two directories: /home/core/tp2-folder2 and /home/core/tp2-folder3. Each directory status page includes a table of files with columns for Path, What it is?, Size (kb), Last Modified, and Synchronize?.

**Status of directory [/home/core/tp2-folder2](#)**

**List of files**

Path	What it is?	Size (kb)	Last Modified	Synchronize?
<a href="#">CC-Topo-2022.imm</a>	FILE	29	Sun Oct 10 11:40:54 WEST 2021	TRUE
<a href="#">Chapter_3_v8.0.pptx</a>	FILE	5997	Sun Oct 24 19:43:32 WEST 2021	TRUE
<a href="#">topo.imm</a>	FILE	29	Sun Oct 24 19:43:32 WEST 2021	TRUE
<a href="#">tp2-folder1</a>	DIRECTORY	4	Thu Dec 23 13:20:55 WET 2021	FALSE
<a href="#">wireshark.tar.xz</a>	FILE	31577	Wed Oct 06 20:16:46 WEST 2021	TRUE
<a href="#">bootstrap-dist.zip</a>	FILE	578	Thu Jan 18 18:09:19 WET 2018	TRUE
<a href="#">book.jpg</a>	FILE	184	Fri Jul 30 17:33:35 WEST 2021	TRUE
<a href="#">rfc7231.txt</a>	FILE	229	Sun Oct 24 19:45:00 WEST 2021	TRUE

**Status of directory [/home/core/tp2-folder3](#)**

**List of files**

Path	What it is?	Size (kb)	Last Modified	Synchronize?
<a href="#">CC-Topo-2022.imm</a>	FILE	29	Sun Oct 10 11:40:54 WEST 2021	TRUE
<a href="#">Chapter_3_v8.0.pptx</a>	FILE	5997	Sun Oct 24 19:43:32 WEST 2021	TRUE
<a href="#">01a.txt</a>	FILE	32	Thu Dec 23 13:22:33 WET 2021	FALSE
<a href="#">topo.imm</a>	FILE	29	Sun Oct 24 19:43:32 WEST 2021	TRUE
<a href="#">wireshark.tar.xz</a>	FILE	31577	Wed Oct 06 20:16:46 WEST 2021	TRUE
<a href="#">bootstrap-dist.zip</a>	FILE	578	Thu Jan 18 18:09:19 WET 2018	TRUE
<a href="#">book.jpg</a>	FILE	184	Fri Jul 30 17:33:35 WEST 2021	TRUE
<a href="#">rfc7231.txt</a>	FILE	229	Sun Oct 24 19:45:00 WEST 2021	TRUE

Fig. 3: Pedido http à página principal com pastas por sincronizar

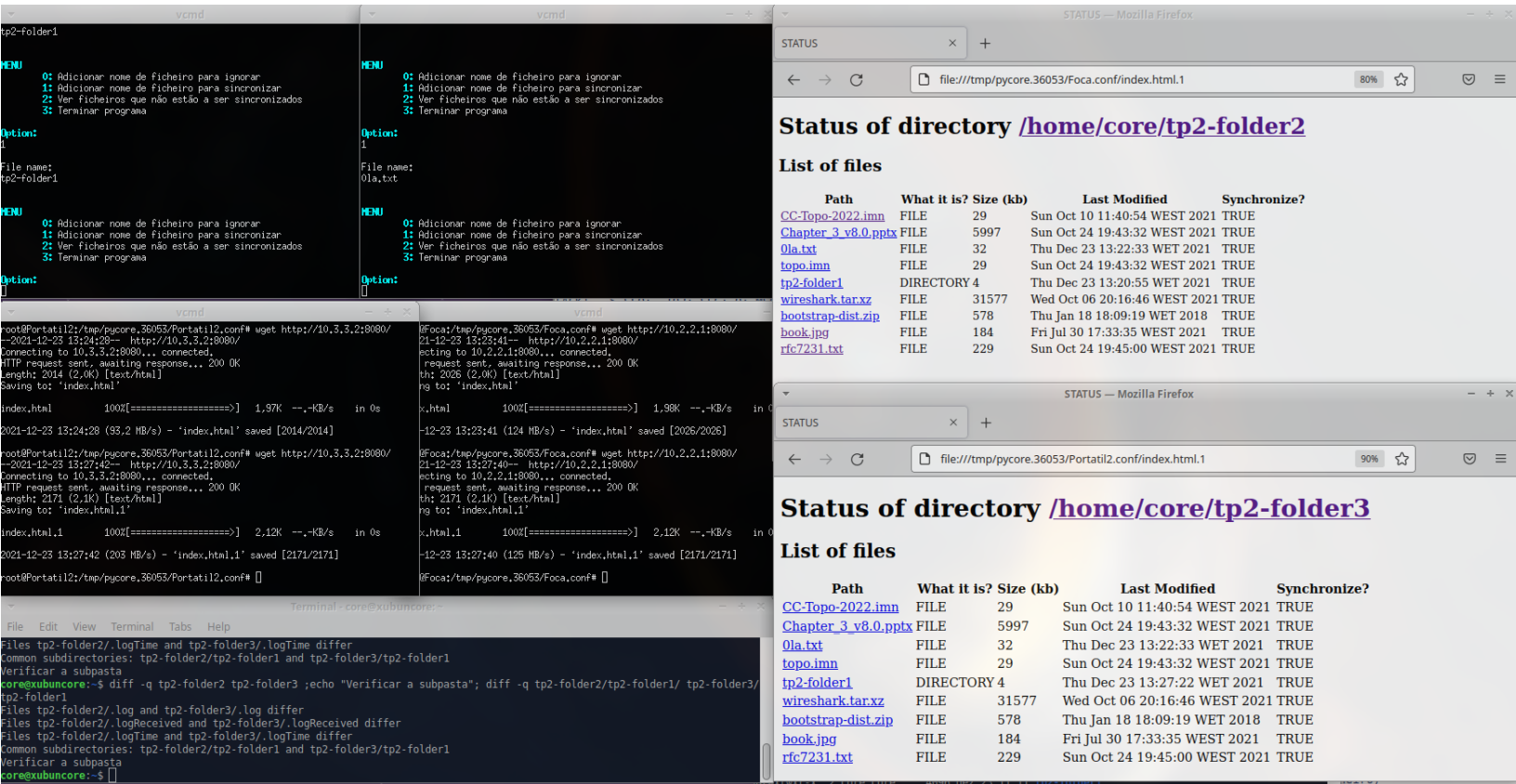
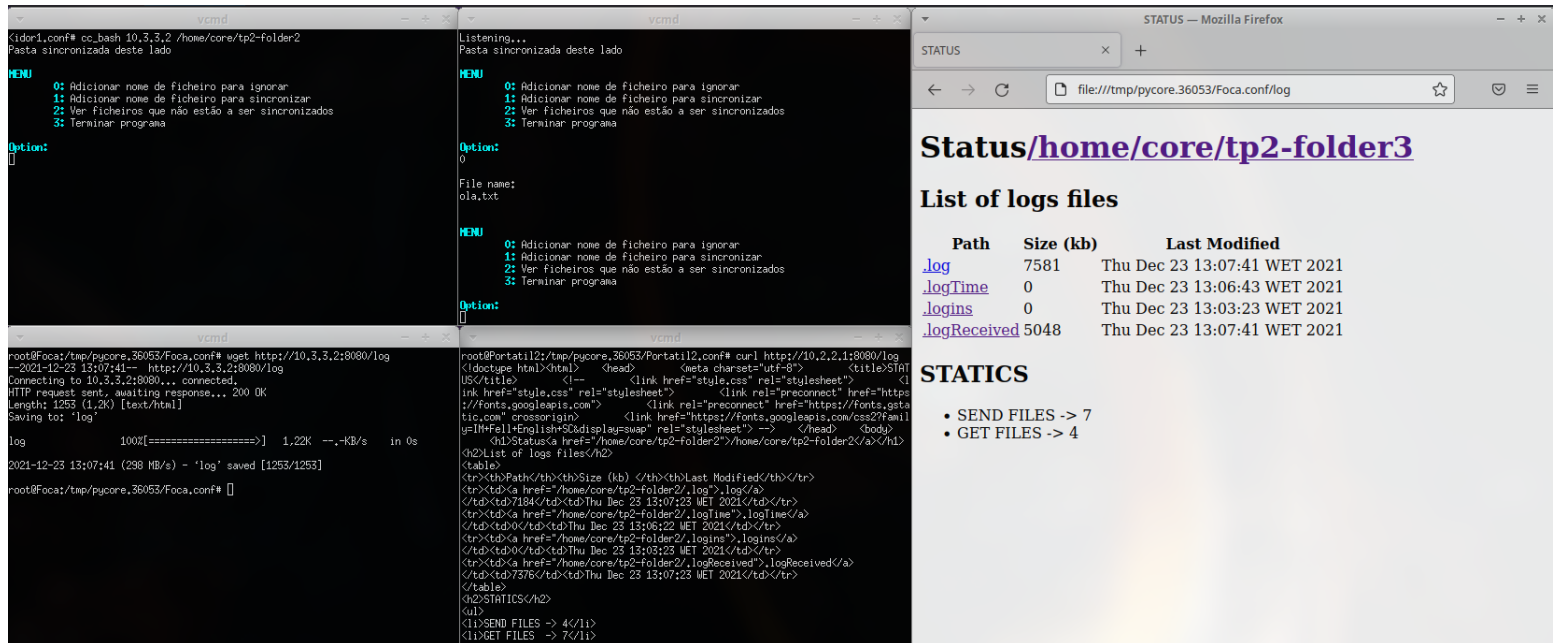


Fig. 4: Pedido http à página principal adicionando pastas a sincronizar

## Trabalho Prático 2 Comunicação por Computadores PL6 Grupo 66



The screenshot shows a web application interface with a terminal window on the left and a main content area on the right. The terminal window displays a menu with options to add file names, synchronize files, and terminate the program. The main content area shows the status of log files and a table of static files.

**Status/home/core/tp2-folder3**

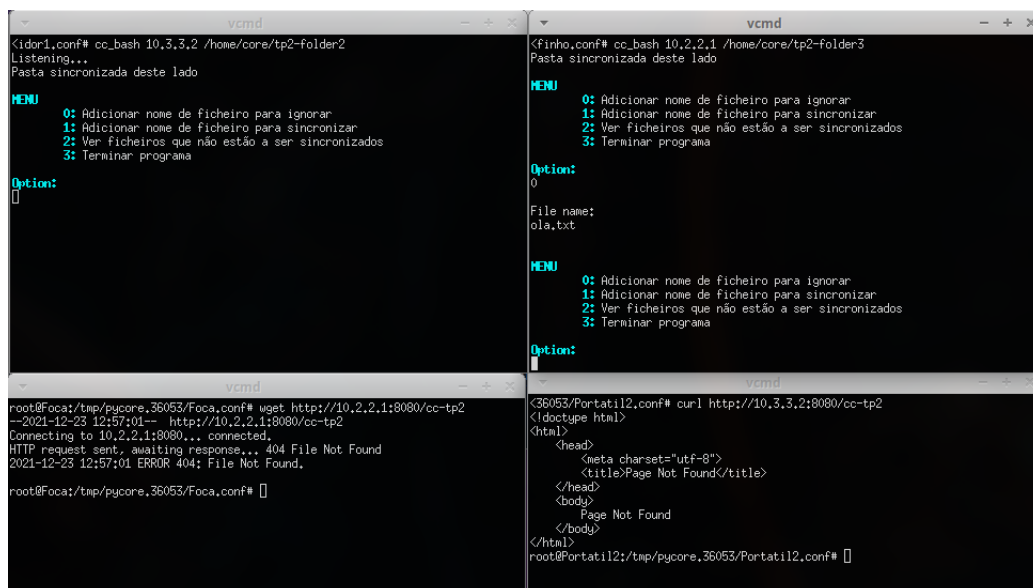
**List of logs files**

Path	Size (kb)	Last Modified
<a href="#">log</a>	7581	Thu Dec 23 13:07:41 WET 2021
<a href="#">logTime</a>	0	Thu Dec 23 13:06:43 WET 2021
<a href="#">logins</a>	0	Thu Dec 23 13:03:23 WET 2021
<a href="#">logReceived</a>	5048	Thu Dec 23 13:07:41 WET 2021

**STATICS**

- SEND FILES -> 7
- GET FILES -> 4

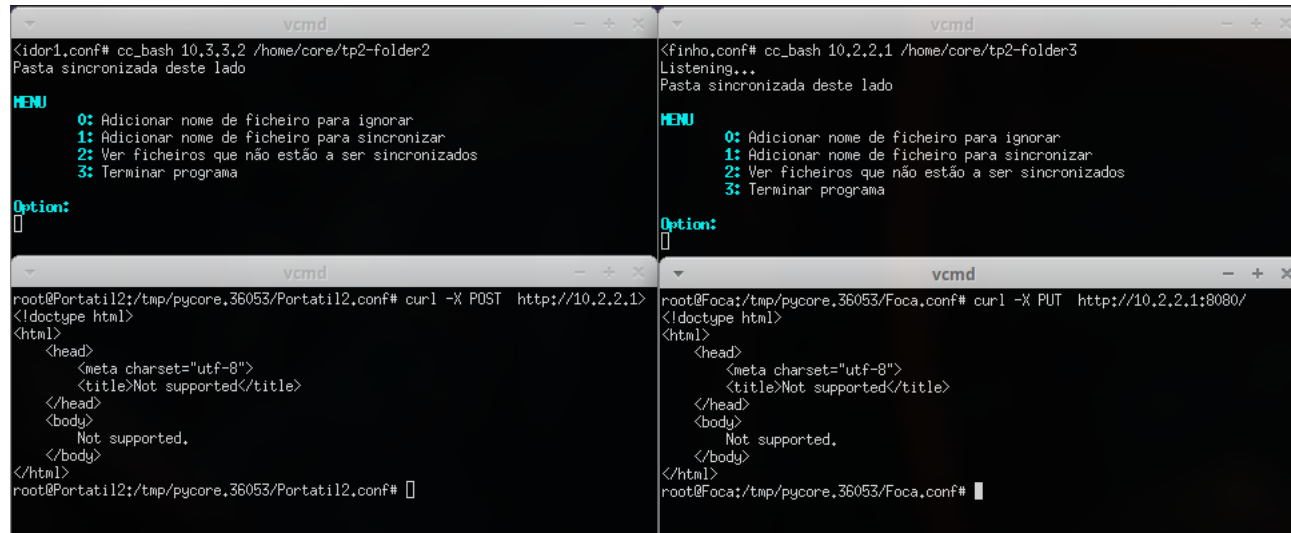
Fig. 5: Pedido http à página dos log



The screenshot shows a web application interface with a terminal window on the left and a main content area on the right. The terminal window displays a menu with options to add file names, synchronize files, and terminate the program. The main content area shows a 'Page Not Found' error message.

**Page Not Found**

Fig. 6: Pedidos http : Page Not Found



The figure displays four terminal windows arranged in a 2x2 grid, all titled 'vcmd'. The top-left window shows a user at 'idor1.conf' running 'cc\_bash 10,3,3,2 /home/core/tp2-folder2', followed by 'Pasta sincronizada deste lado' and a menu with options 0 (Adicionar nome de ficheiro para ignorar), 1 (Adicionar nome de ficheiro para sincronizar), 2 (Ver ficheiros que não estão a ser sincronizados), and 3 (Terminar programa). The user has selected option 0. The top-right window shows a user at 'finho.conf' running 'cc\_bash 10,2,2,1 /home/core/tp2-folder3', followed by 'Listening...' and 'Pasta sincronizada deste lado', with the same menu. The bottom-left window shows a user at 'root@Portatil12:/tmp/pycore,36053/Portatil12.conf' running 'curl -X POST http://10.2.2.1', resulting in an HTML response with a title 'Not supported'. The bottom-right window shows a user at 'root@Foca:/tmp/pycore,36053/Foca.conf' running 'curl -X PUT http://10.2.2.1:8080/', also resulting in an HTML response with a title 'Not supported'.

```
<idor1.conf# cc_bash 10,3,3,2 /home/core/tp2-folder2
Pasta sincronizada deste lado

MENU
0: Adicionar nome de ficheiro para ignorar
1: Adicionar nome de ficheiro para sincronizar
2: Ver ficheiros que não estão a ser sincronizados
3: Terminar programa

Option:
0

<finho.conf# cc_bash 10,2,2,1 /home/core/tp2-folder3
Listening...
Pasta sincronizada deste lado

MENU
0: Adicionar nome de ficheiro para ignorar
1: Adicionar nome de ficheiro para sincronizar
2: Ver ficheiros que não estão a ser sincronizados
3: Terminar programa

Option:
0

root@Portatil12:/tmp/pycore,36053/Portatil12.conf# curl -X POST http://10.2.2.1/
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Not supported</title>
  </head>
  <body>
    Not supported.
  </body>
</html>
root@Portatil12:/tmp/pycore,36053/Portatil12.conf#

root@Foca:/tmp/pycore,36053/Foca.conf# curl -X PUT http://10.2.2.1:8080/
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Not supported</title>
  </head>
  <body>
    Not supported.
  </body>
</html>
root@Foca:/tmp/pycore,36053/Foca.conf#
```

Fig. 7: Pedidos http : Error 501 Not Implemented

## Testes

Aqui, é de salientar que, visto existirem os ficheiros de *log* e de segurança em cada uma das pastas a sincronizar e sendo estes diferentes e únicos para cada pasta, estes ficheiros não serão sincronizados, pelo que, quando se testar com o comando *diff*, se as pastas se encontram iguais, estes ficheiros serão diferentes.

### A Sincronização da pasta tp2-folder3 com a tp2-folder2

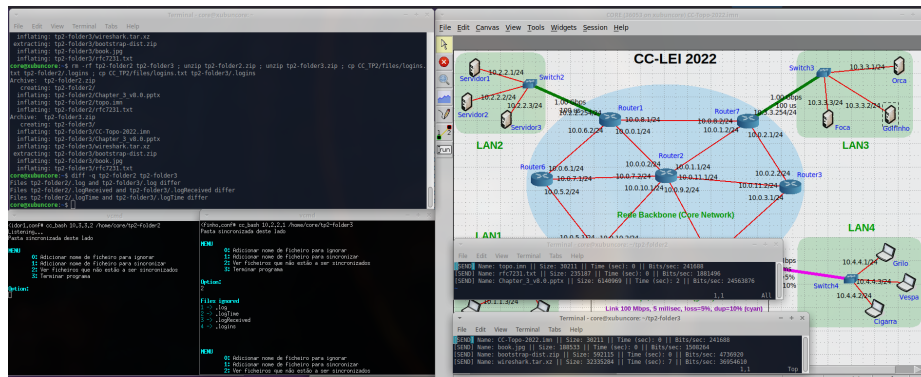


Fig. 8: Sincronização da pasta tp2-folder3 com a tp2-folder2

## B Sincronização de subpastas

Aqui testamos a sincronização da pasta `tp2-folder2` com a pasta `tp2-folder3`, esta mesma conterá a pasta `tp2-folder2`.

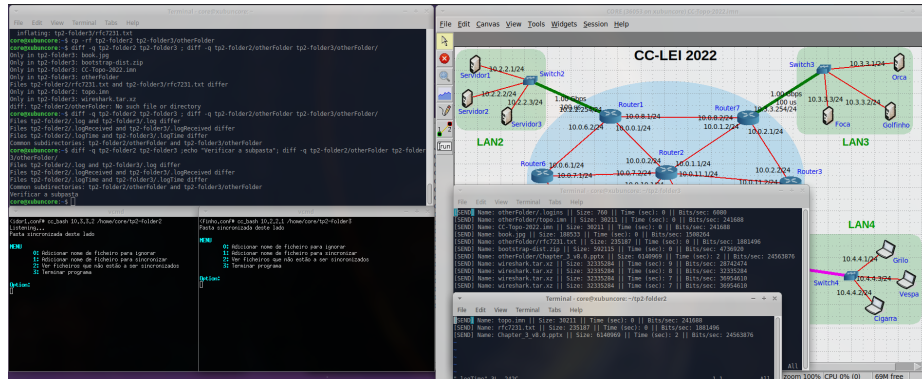
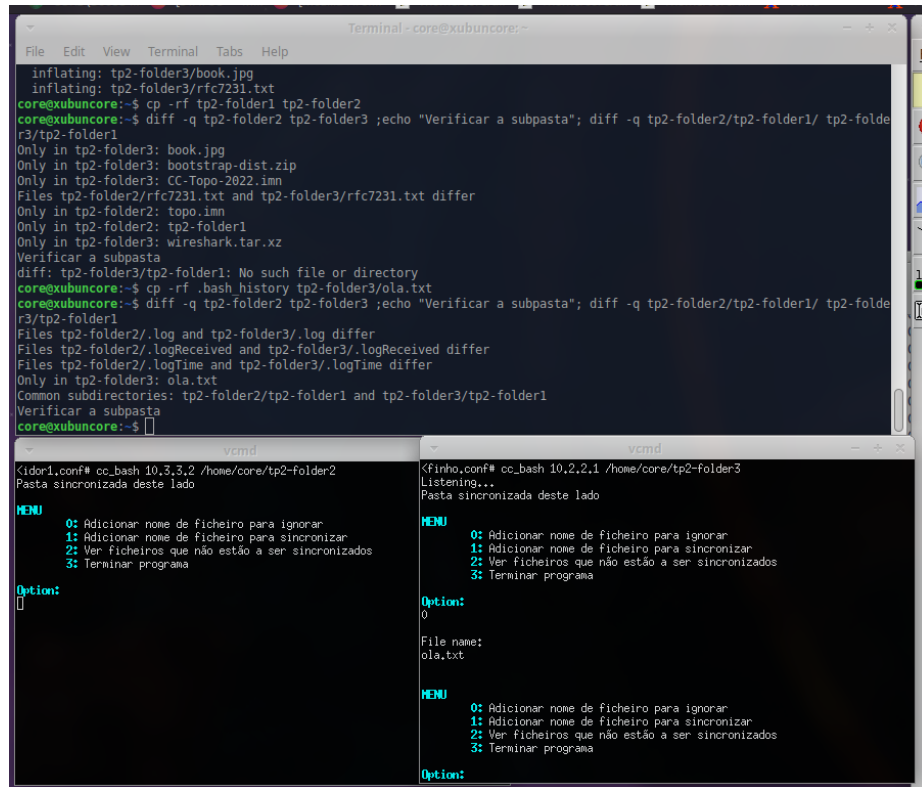


Fig. 9: Sincronização da pasta `tp2-folder3` que contém a pasta `tp2-folder2` com a `tp2-folder2`

Aqui, testamos a sincronização da pasta `tp2-folder2` com a pasta `tp2-folder3`, esta mesma conterá a pasta `tp2-folder1`. Além disso, adicionámos a `tp2-folder3` o ficheiro `ola.txt`, ficheiro este adicionado aos ficheiros a não serem sincronizados.



## Trabalho Prático 2 Comunicação por Computadores PL6 Grupo 66



```
Terminal - core@xubuncore:~  
File Edit View Terminal Tabs Help  
inflating: tp2-folder3/book.jpg  
inflating: tp2-folder3/rfc7231.txt  
core@xubuncore:~$ cp -rf tp2-folder1 tp2-folder2  
core@xubuncore:~$ diff -q tp2-folder2 tp2-folder3 ;echo "Verificar a subpasta"; diff -q tp2-folder2/tp2-folder1/ tp2-fold  
r3/tp2-folder1  
Only in tp2-folder3: book.jpg  
Only in tp2-folder3: bootstrap-dist.zip  
Only in tp2-folder3: CC-Topo-2022.imn  
Files tp2-folder2/rfc7231.txt and tp2-folder3/rfc7231.txt differ  
Only in tp2-folder2: topo.imn  
Only in tp2-folder2: tp2-folder1  
Only in tp2-folder3: wireshark.tar.xz  
Verificar a subpasta  
diff: tp2-folder3/tp2-folder1: No such file or directory  
core@xubuncore:~$ cp -rf .bash_history tp2-folder3/ola.txt  
core@xubuncore:~$ diff -q tp2-folder2 tp2-folder3 ;echo "Verificar a subpasta"; diff -q tp2-folder2/tp2-folder1/ tp2-fold  
r3/tp2-folder1  
Files tp2-folder2/.log and tp2-folder3/.log differ  
Files tp2-folder2/.logReceived and tp2-folder3/.logReceived differ  
Files tp2-folder2/.logTime and tp2-folder3/.logTime differ  
Only in tp2-folder3: ola.txt  
Common subdirectories: tp2-folder2/tp2-folder1 and tp2-folder3/tp2-folder1  
Verificar a subpasta  
core@xubuncore:~$
```

```
vcmid  
<ldor1.conf# cc_bash 10.3.3.2 /home/core/tp2-folder2  
Pasta sincronizada deste lado  
MENU  
0: Adicionar nome de ficheiro para ignorar  
1: Adicionar nome de ficheiro para sincronizar  
2: Ver ficheiros que não estão a ser sincronizados  
3: Terminar programa  
Option:  
0
```

```
vcmid  
<finho.conf# cc_bash 10.2.2.1 /home/core/tp2-folder3  
Listening...  
Pasta sincronizada deste lado  
MENU  
0: Adicionar nome de ficheiro para ignorar  
1: Adicionar nome de ficheiro para sincronizar  
2: Ver ficheiros que não estão a ser sincronizados  
3: Terminar programa  
Option:  
0  
File name:  
ola.txt  
MENU  
0: Adicionar nome de ficheiro para ignorar  
1: Adicionar nome de ficheiro para sincronizar  
2: Ver ficheiros que não estão a ser sincronizados  
3: Terminar programa  
Option:
```

Fig. 10: Sincronização de pastas com ficheiros a não serem sincronizados

Posteriormente, voltou-se a adicionar o ficheiro *ola.txt* aos ficheiros a serem sincronizados.

```
Terminal - core@xubuncore:~
File Edit View Terminal Tabs Help
Only in tp2-folder3: CC-Topo-2022.imn
Files tp2-folder2/rfc7231.txt and tp2-folder3/rfc7231.txt differ
Only in tp2-folder2: topo.imn
Only in tp2-folder2: tp2-folder1
Only in tp2-folder3: wireshark.tar.xz
Verificar a subpasta
diff: tp2-folder3/tp2-folder1: No such file or directory
core@xubuncore:~$ cp -rf .bash_history tp2-folder3/ola.txt
core@xubuncore:~$ diff -q tp2-folder2 tp2-folder3 ;echo "Verificar a subpasta"; diff -q tp2-folder2/tp2-folder1/ tp2-folde
r3/tp2-folder1
Files tp2-folder2/.log and tp2-folder3/.log differ
Files tp2-folder2/.logReceived and tp2-folder3/.logReceived differ
Files tp2-folder2/.logTime and tp2-folder3/.logTime differ
Only in tp2-folder3: ola.txt
Common subdirectories: tp2-folder2/tp2-folder1 and tp2-folder3/tp2-folder1
Verificar a subpasta
core@xubuncore:~$ diff -q tp2-folder2 tp2-folder3 ;echo "Verificar a subpasta"; diff -q tp2-folder2/tp2-folder1/ tp2-folde
r3/tp2-folder1
Files tp2-folder2/.log and tp2-folder3/.log differ
Files tp2-folder2/.logReceived and tp2-folder3/.logReceived differ
Files tp2-folder2/.logTime and tp2-folder3/.logTime differ
Common subdirectories: tp2-folder2/tp2-folder1 and tp2-folder3/tp2-folder1
Verificar a subpasta
core@xubuncore:~$
```

```
vcmd
kldor1.conf# cc_bash 10.3.3.2 /home/core/tp2-folder2
Pasta sincronizada deste lado
MENU:
0: Adicionar nome de ficheiro para ignorar
1: Adicionar nome de ficheiro para sincronizar
2: Ver ficheiros que não estão a ser sincronizados
3: Terminar programa
Option:
1
File name:
ola.txt
```

Fig. 11: Sincronização de pastas após ser adicionado um ficheiro aos ficheiros sincronizados