



Controlo e Edição

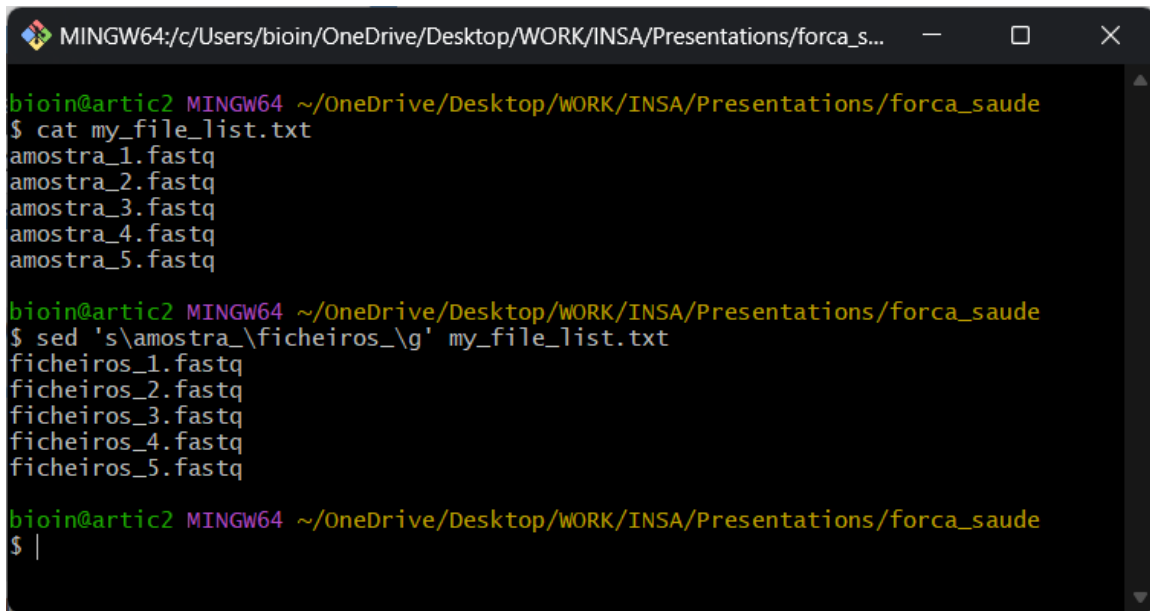
SED - editar em uma linha

Sed é um commando poderoso que permite manipular texto dentro de um ficheiro.

```
sed [options] [filename]
```

O mais frequente é utiliza lo para substituir padroes. O formato é :

```
sed 's\[padrao a modificar\]\[padrão de substituicao\]g' [nome do ficheiro]
```



```
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ cat my_file_list.txt
amostra_1.fastq
amostra_2.fastq
amostra_3.fastq
amostra_4.fastq
amostra_5.fastq

bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ sed 's\amostra_\ficheiros_g' my_file_list.txt
ficheiros_1.fastq
ficheiros_2.fastq
ficheiros_3.fastq
ficheiros_4.fastq
ficheiros_5.fastq

bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ |
```

Agora vimos como modificar o que lemos de um ficheiro. mas notem que nao modificamos o seu conteudo! se abrirmos o ficheiro continua igual.

- com o argumento `sed -i` modificamos o ficheiro sem escrever para a shell.

Mas e entao e se, em vez de moficar o ficheiro, nós quisessemos escrever um ficheiro novo, com o resultado da nossa modificacao?

Pipes e redirecoes.

Duas das caracteristicas mais poderosas da shell.

Primeiro precisamos de rever uns conceitos:

- Input : `STDIN 0` até agora, vimos que o input de um comando da shell é o ficheiro que vem no fim. no problem!

- Output : `STDOUT 1` até agora vimos o output a aparecer escrito na SHELL. Este é o **Standard Output**, e é o comportamento default, mas não o unico!
- Erro: `STDERR 2` este nós ainda nao vimos, mas corresponde a msgs de erro quando o programa nao funciona.

Estes são os tres canais de mensagem que sao respeitados por todos os comandos UNIX.

exemplo:

```
### STDIN neste caso é o cadeado
### STDOUT é a SHELL!
echo "hello world"
```

um exemplo de STDERR:

```
ls oienrgvoiaenrg ## lista de elementos, mas a directoria que na
```

Controlar o Output

Em UNIX, controlamos a saída de um programa usando o simbolo `>`

- STDOUT - `>` ou `1>`

```
echo "hello world" > my_file.txt
```

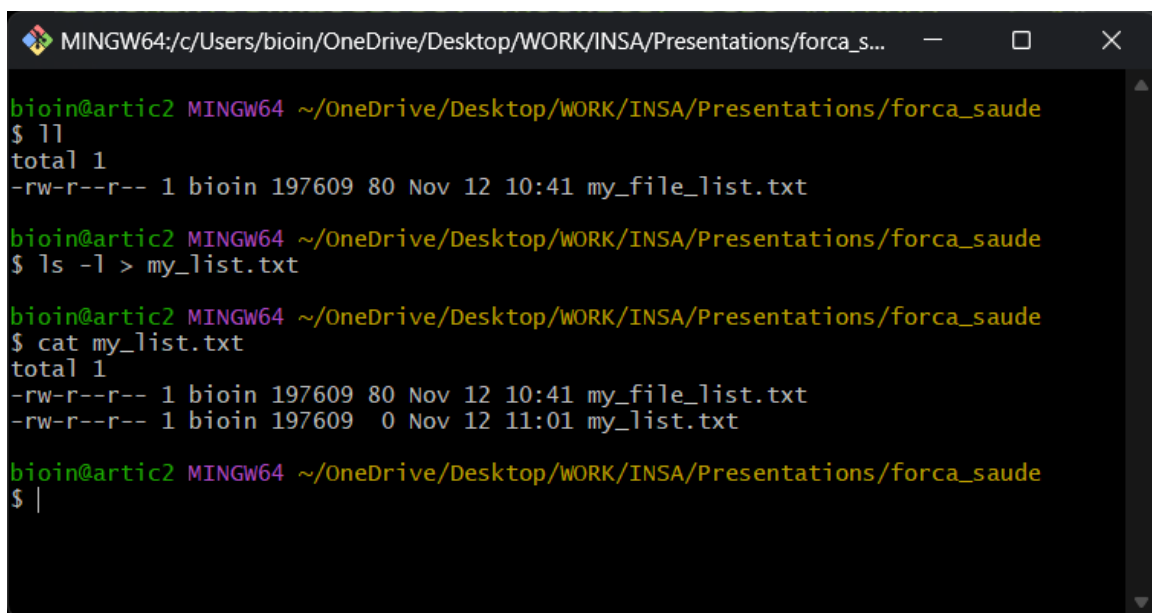
- STERR - `2>`

```
cat hello 2> my_error_output.txt
```

Se utilizarmos o símbolo `>` apenas uma vez, o nosso comando substitui qualquer conteúdo do ficheiro de destino!

para adicionar linhas novas ao nosso ficheiro de destino, usamos `>>`

O output de qualquer comando pode ser redirecionado desta maneira.



```
MINGW64:/c:/Users/bioin/OneDrive/Desktop/WORK/INSA/Presentations/forca_s...
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ ll
total 1
-rw-r--r-- 1 bioin 197609 80 Nov 12 10:41 my_file_list.txt
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ ls -l > my_list.txt
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ cat my_list.txt
total 1
-rw-r--r-- 1 bioin 197609 80 Nov 12 10:41 my_file_list.txt
-rw-r--r-- 1 bioin 197609  0 Nov 12 11:01 my_list.txt
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ |
```

Pipes

Vimos como controlar o output de um programa. mas também é possível controlar o input!

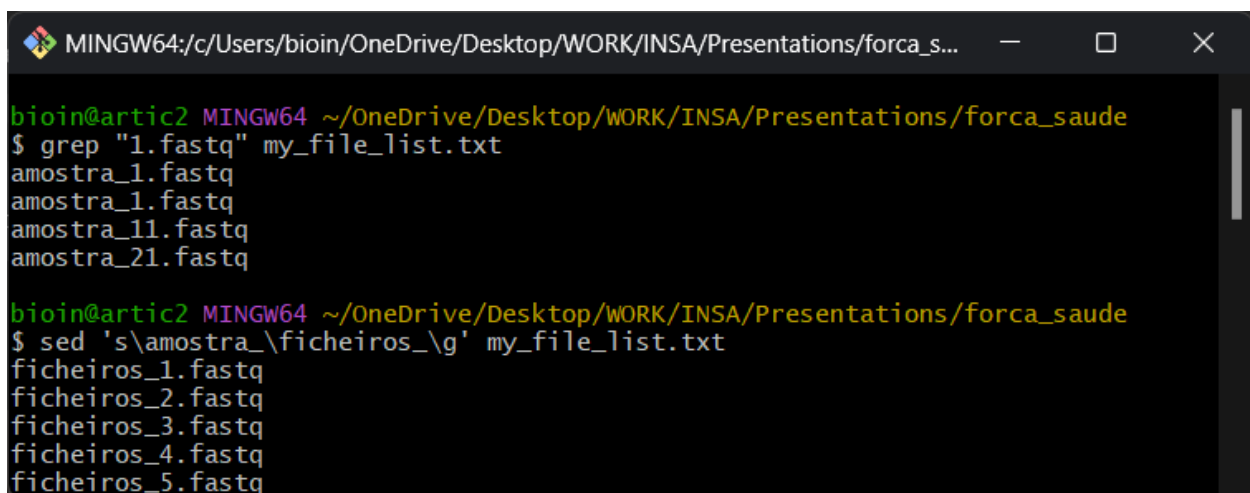
Todos os comandos de UNIX, e muitos programas em bioinformática, estão preparados para que o INPUT seja passado “implicitamente”. O que é que isto quer dizer?

Vamos ver um exemplo:

Imaginemos que a partir da minha lista de ficheiros, quero :

- apenas as amostras cujo nome acabe em “1”, e
- para estas quero trocar “amostra” por “ficheiro”.

Eu posso fazer os dois separadamente:



```
MINGW64:/c/Users/bioin/OneDrive/Desktop/WORK/INSA/Presentations/forca_s...  
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude  
$ grep "1.fastq" my_file_list.txt  
amostra_1.fastq  
amostra_1.fastq  
amostra_11.fastq  
amostra_21.fastq  
  
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude  
$ sed 's\amostra\ficheiros\g' my_file_list.txt  
ficheiros_1.fastq  
ficheiros_2.fastq  
ficheiros_3.fastq  
ficheiros_4.fastq  
ficheiros_5.fastq
```

Como é que os Junto? com um PIPE, representado pelo símbolo `|`. Com PIPES, comandos UNIX deixam de precisar que o ficheiro de input seja explícito:

```
MINGW64:/c/Users/bioin/OneDrive/Desktop/WORK/INSA/Presentations/forca_s...  
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude  
$ grep "1.fastq" my_file_list.txt | sed 's\amostra\ficheiro\g'  
ficheiro_1.fastq  
ficheiro_1.fastq  
ficheiro_11.fastq  
ficheiro_21.fastq  
  
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude  
$
```

podemos juntar tantos PIPES quanto quisermos:

```
MINGW64:/c/Users/bioin/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude  
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude  
$ grep "1.fastq" my_file_list.txt | sed 's\amostra\ficheiro\g' | sed 's\fastq\fastq.gz\g'  
ficheiro_1.fastq.gz  
ficheiro_1.fastq.gz  
ficheiro_11.fastq.gz  
ficheiro_21.fastq.gz  
  
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude  
$
```

No final de um pipe, podemos redirecionar o nosso output para um ficheiro, assim combinando os dois:

```
MINGW64:/c/Users/bioin/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ grep "1.fastq" my_file_list.txt | sed 's\amostra\ficheiro\g' | sed 's\fastq\fastq.gz\g' > output.txt

bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ cat output.txt
ficheiro_1.fastq.gz
ficheiro_1.fastq.gz
ficheiro_11.fastq.gz
ficheiro_21.fastq.gz

bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$
```

Exercício

Criar uma nova lista de ficheiros, em que os nomes de amostra apresentao agora o código "amostra_UTC_" antes do numero atribuido.

Programação Bash

Vamos imaginar que temos uma operacao deste tipo que precisa de ser repetida muitas vezes. por exemplo. em vez de escrever muitas vezes o mesmo, devemos escrever um **script bash**.

- extensão `.sh`
- Para executar um programa a partir da linha de comando, usa se a forma `.[nome do ficheiro]`

comecamos o script por indicar o programa que deve correr este script. este vais ser reconhecido pelo sistema operativo.

```
#!/usr/bin/env bash
#

echo "Hello World"
```

- `#!/bin/bash` tb é corrente, os dois são indicativos do programa bash, que executa este script.

Loops

Muitas vezes, ao escrever um programa (ou na SHELL), é necessário repetir uma operação para diversos ficheiros ou directorias. qualquer coisa do género: “para cada ficheiro”

A programação em `bash` permite este tipo de operações utilizando o formato `for`.

```
#!/bin/bash
# Basic for loop
names='Stan Kyle Cartman Kenny' # is one way to define lists
for name in $names
do
    echo $name
done
echo All done
```

- notem o `do` e o `done` que determinam o princípio e o fim do conteúdo da iteração.
- notem a variável `names` que guarda a lista de nomes. variáveis são muito úteis em bash.
 - quando criamos uma variável em bash, podemos voltar a usar a variável mediante o símbolo `$`.

```
name="JOAO"
echo $name
```

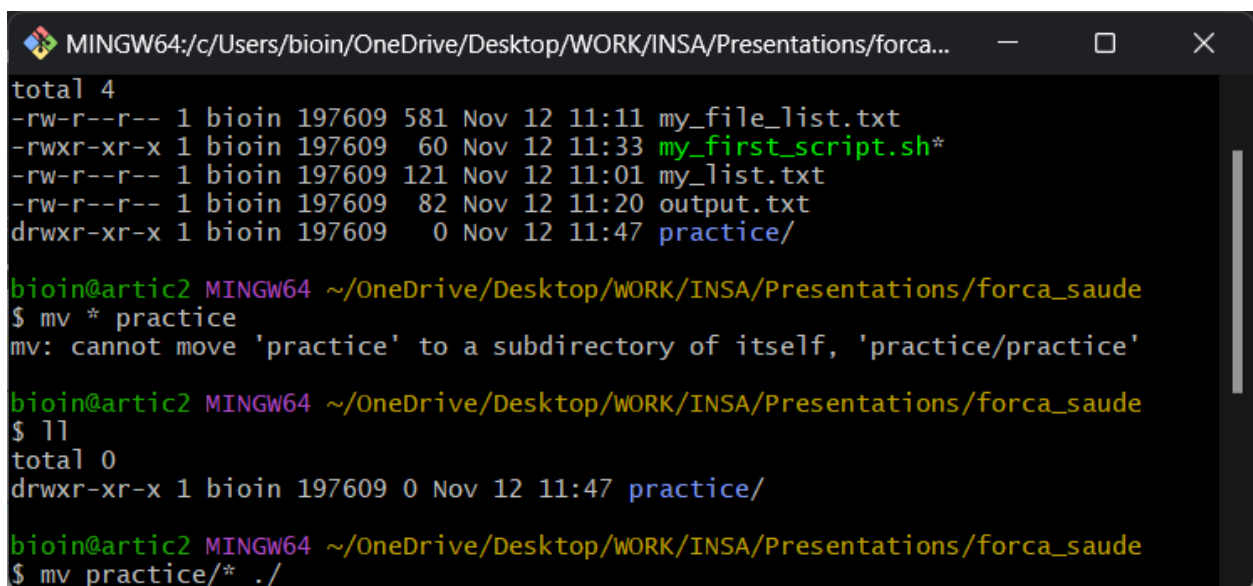

Uma coisa muito frequente é querer criar numeros num faixa (i.e. de 1 a 20 por exemplo).

Neste script, utilizamos um loop para criar directorias de acordo com o padrão.

```
#!/bin/bash
# Basic range in for loop

for value in {1..5}
do
    mkdir "my_basic_directory_number_"$value
done
echo All done
```

num loop, é frequente termos varios ficheiros ou directorias, com padroes repetidos nos nomes, sobre as quais queremos trabalhar, nesse caso, a wildcard `*` é muito util!



```
MINGW64:/c/Users/bioin/OneDrive/Desktop/WORK/INSA/Presentations/forca...
total 4
-rw-r--r-- 1 bioin 197609 581 Nov 12 11:11 my_file_list.txt
-rwxr-xr-x 1 bioin 197609 60 Nov 12 11:33 my_first_script.sh*
-rw-r--r-- 1 bioin 197609 121 Nov 12 11:01 my_list.txt
-rw-r--r-- 1 bioin 197609 82 Nov 12 11:20 output.txt
drwxr-xr-x 1 bioin 197609 0 Nov 12 11:47 practice/

bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ mv * practice
mv: cannot move 'practice' to a subdirectory of itself, 'practice/practice'

bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ ll
total 0
drwxr-xr-x 1 bioin 197609 0 Nov 12 11:47 practice/

bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ mv practice/* ./
```

Vamos por exemplo, adicionar um ficheiro de results em cada uma das nossas directorias:

```
MINGW64:/c/Users/bioin/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ ll
total 0
drwxr-xr-x 1 bioin 197609 0 Nov 12 11:48 my_basic_directory_number_1/
drwxr-xr-x 1 bioin 197609 0 Nov 12 11:48 my_basic_directory_number_2/
drwxr-xr-x 1 bioin 197609 0 Nov 12 11:48 my_basic_directory_number_3/
drwxr-xr-x 1 bioin 197609 0 Nov 12 11:48 my_basic_directory_number_4/
drwxr-xr-x 1 bioin 197609 0 Nov 12 11:48 my_basic_directory_number_5/
drwxr-xr-x 1 bioin 197609 0 Nov 12 11:48 practice/

bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ for dir in my_basic_directory_number_*;
> do
> echo "my analysis" > $dir"/results.txt";
> done

bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$ ll my_basic_directory_number_1/
total 1
-rw-r--r-- 1 bioin 197609 12 Nov 12 11:51 results.txt

bioin@artic2 MINGW64 ~/OneDrive/Desktop/WORK/INSA/Presentations/forca_saude
$
```

Exercicio 1

Escrever um loop para criar uma serie de nomes em série, com números de 1 a 10, seguindo o padrão acima. A cada iteração, imprimir o nome do ficheiro usando "echo"

Exercício 2

entrar na pasta `/UNIX/folders/`. esta pasta contem várias sub-directorias com um ficheiro de resultados, `results.txt`, cada.

1. Escrever um loop para imprimir o nome de cada uma das pastas (ver acima)
2. Escrever um loop para encontrar qual pasta contem a amostra "amostra_23"