



Rede de Instituições de Ensino Superior e de Investigação de Moçambique

(MoRENNet)

**Formação em Computação de Alto Desempenho Edição 2022: Introdução
ao Shell e Bash Scripting**

OpenHPC MoRENNet Edição 2022

Maputo, Outubro de 2022

Direitos de autor (copyright)

Este manual não pode ser reproduzido para fins comerciais. Caso haja necessidade de reprodução, deverá ser mantida a referência ao Instituto Nacional de Governo Electrónico, Instituto Público e aos seus Autores.

OpenHPC MoRENet Edição 2022



Instituto Nacional do Governo Electrónico, Instituto Público

Av. Vladimir Lenine Nº 598, 7º e 8º

Website: <https://www.inage.gov.mz>

Ficha Técnica

Autor: Martílio Rafael Banze

OpenHPC MoRENet Edição 2022

Contents

1. Introdução	5
3. Atribuições do SO	6
4. Divisão do SO.....	6
5. Linux.....	6
5.1. Linux kernel.....	7
5.2. Utilitários associados.....	7
5.3. Distribuições Linux	7
6. Vantagens e Desvantagens (Linux e Windows).....	7
7. Usando um sistema Linux.....	7
7.1. Linux Command Line.....	8
7.2. Logging Out.....	10
7.3. Sintaxe do Comando	10
7.4. Directórios	17
7.5. Flag adicionais e localização de arquivos	24
7.6. Variáveis e Loops	31
7.7. Bash scripting Parte 1.....	34
7.8. Permissions: Octal Representation	37
8. Exemplo de um job_script de Python:	40
9. Como usar o comando scp para transferir arquivos com segurança.....	41
9.1. Copiando arquivos e directórios entre dois sistemas com scp	41
9.2. Para copiar um directório de um sistema local para remoto, use a opção -r:	42
9.3. Copiando um arquivo remoto para um sistema local usando o comando scp	42
9.4. Copiando um arquivo entre dois sistemas remotos usando o comando scp	43

Visão Geral

Para este tópico os participantes do treinamento em HPC aprenderão a trabalhar com sistemas comuns de computação de alto desempenho. Isso inclui navegar em sistemas de arquivos, trabalhar com um sistema operacional HPC típico (Linux).

OpenHPC MoRENet Edição 2022

1. Introdução

Trabalhando com este guia (parte 1) na linha de comando do Linux (Bash), você vai adquirir técnicas, dicas e truques poderosos para tornar sua vida mais fácil em pouco tempo. As páginas a seguir destinam-se a fornecer uma base sólida sobre como usar o terminal, para que o computador faça um trabalho útil para você.

Aqui você aprenderá a linha de comando do Linux (Bash) com nosso tutorial elaborado especificamente para o treinamento em HPC. Ele contém descrições claras, linhas de comando, exemplos, atalhos e melhores práticas.

A princípio, a linha de comando do Linux pode parecer complexa e assustadora. Na verdade, é bastante simples e intuitivo (uma vez que você entenda o que está acontecendo) e, uma vez que você trabalhe nas seções a seguir, você entenderá o que está acontecendo.

Uma pergunta que pode ter passado pela sua cabeça é: "Por que devo preocupar-me em aprender a linha de comando? A Interface Gráfica do Usuário é muito mais fácil e já consigo fazer a maior parte do que preciso lá."

Até certo ponto, você estaria certo e, de forma alguma, estou sugerindo que você abandone a [GUI](#). Algumas tarefas são mais adequadas para uma [GUI](#), processamento de texto e edição de vídeo são ótimos exemplos. Ao mesmo tempo, algumas tarefas são mais adequadas à linha de comando, manipulação de dados (relatórios) e gerenciamento de arquivos são alguns bons exemplos. Algumas tarefas serão igualmente fáceis em qualquer ambiente. Pense na linha de comando como outra ferramenta que você pode adicionar ao seu cinto. Como sempre, escolha a melhor ferramenta para o trabalho. Por exemplo, no nosso Cluster, em todos os nodos corre [Linux\(CentOS7\)](#) e usa o *shell Bash* por padrão. Os objectivos que nos levam a aprender o linux são:

- Navegar em um ambiente típico de HPC baseado em Linux;
- Demonstrar como executar comandos em um script bash
- Criar novos scripts do Bash.
- Discutir os diferentes locais de armazenamento de arquivos em um sistema HPC;
- Trabalhar com sistemas remotos.

1. Sistema Operacional (SO)

OS é um programa cuja principal função é servir de interface entre um computador e usuário.

2. Atribuições do SO

- Gerenciamento de processos;
- Gerenciamento de memória;
- Controlo do sistema de arquivo;
- Controlo do fluxo de dados;
- Gerenciamento do conjunto de Hardwares e Softwares.

Nota: Sistema de arquivos é um conjunto de estruturas lógicas e de rotinas que permitem ao sistema operacional controlar o acesso ao disco rígido.

3. Divisão do SO

- 1) **Kernel**- o núcleo do SO. O núcleo do Linux é o componente central do sistema, ele serve de ponte entre aplicativos e o processamento real de dados feitos a nível de hardware.

As responsabilidades do núcleo incluem gerenciar os recursos do sistema: comunicação entre hardware e software. Forma a estrutura base do sistema operacional.

- 2) **Shell**-o interpretador de comandos.

4. Linux

Linux é um Sistema Operacional, assim como o Windows e o Mac OS, que possibilita a execução de programas em um computador e outros dispositivos. Linux foi baseado no Unix (vide a Figura 1). Linux pode ser livremente modificado e distribuído.

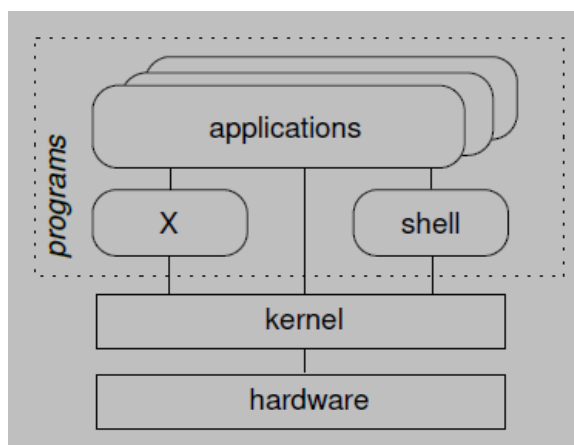


Figura 1: Arquitectura do sistema Unix

4.1. Linux kernel

- Desenvolvido por Linus Torvalds.
- Estritamente falando, 'Linux' é apenas o kernel.

4.2. Utilitários associados

- Ferramentas padrão encontradas em (quase) todos os sistemas Linux
- Muitas partes importantes vêm do projecto [GNU](#).
 - O projecto da Free Software Foundation para fazer um Unix gratuito
 - Alguns afirmam que o sistema operacional como um todo deve ser 'GNU/Linux'

4.3. Distribuições Linux

- Kernel mais utilitários e outras ferramentas, empacotados para usuários finais
- Geralmente com programa de instalação
- Os distribuidores incluem: Red Hat, Debian, SuSE, Mandrake, etc

5. Vantagens e Desvantagens (Linux e Windows)

Por que usamos Linux na Infra-estrutura de HPC?

Em todos os nós do cluster SISCAD-A corre CentOS do Linux, versão 7 por ser uma distribuição gratuita, baseada na distribuição licenciada do Red Enterprise Linux, com maturidade e estabilidade suficientes para garantir um funcionamento adequado e sem falhas, isto é, o SO Linux não necessita de muita memória para sua operação e conforme os programas vão sendo abertos, mais memória vai sendo alocada de forma mais eficiente. O sistema lida bem em casos de sobras de memória, utilizando os MBytes dos módulos como cache de discos.

Cache de discos são porções de memória RAM usadas por arquivos e bibliotecas lidos do HD que têm uma maior probabilidade de serem acessados, uma espécie de Prefetch, o que melhora o desempenho do sistema.

O uso do Linux oferece como grande vantagem a **possibilidade de compilar o núcleo de modo personalizado**, isto é, apenas com os módulos específicos para o hardware dos nós computacionais. Além da economia de espaço em memória para as aplicações, permite uma maior estabilidade do sistema operacional, pela redução do número de componentes envolvidos.

6. Usando um sistema Linux

Tabela 1: Resumo dos comandos usados

Comandos	Resultado
cal	Dá-lhe uma vista do calendário no terminal
date	Data e hora (actuais)
man	Exibe manual para um comando específico do Linux

clear	Limpa o que foi escrito e devolvido por kernel no terminal
pwd	"Present Working Directory"
ls	Faz listagem do que está no presente no directório actual
echo	Imprimir texto na tela
df	Verifica a quantidade de espaço livre em disco
free	Verifica a memória em uma máquina

O terminal é uma forma de comunicação interactiva com o computador. Você digita um chamado "comando" e, em seguida, pressiona a tecla Return (Enter) para enviar sua instrução. Isso faz com que o bash, o interpretador padrão, processe seu comando e provavelmente execute um programa. Você então vê a saída desse programa como resultado. Depois que terminar, você verá o prompt novamente e poderá digitar o próximo comando.

Você pode interromper um programa travado pressionando Ctrl+c: isso envia um sinal de "interrupção" para o processo em execução.

6.1. Linux Command Line

- O shell é onde os comandos são invocados
- Um comando é digitado em um prompt de Shell
 - O prompt geralmente termina em um símbolo (\$)
- Depois de digitar um comando, pressione "Enter" para invocá-lo
 - O shell tentará obedecer ao comando
 - Outro prompt aparecerá

Exemplos

- Agora vamos exibir o calendário e a data e hora atuais:

```
[mbanze@strge ~]$ date
Fri Jul 1 14:56:11 CAT 2022
[mbanze@strge ~]$
```

```
[mbanze@strge scratch]$ cal
      July 2022
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
[mbanze@strge scratch]$
[mbanze@strge scratch]$ date
Thu Jul 7 11:17:28 CAT 2022
[mbanze@strge scratch]$
```

- O dolar sign representa o prompt neste curso — não o digite.
- Caso deseje remover o conteúdo exibido na tela actual, você pode usar: *clear*
 - *clear* simplesmente limpa a tela para que possamos trabalhar em uma nova tela. Não se preocupe, os comandos que você digitou até agora não serão perdidos. Mostrarei, mais adiante no curso, como você pode recuperar todos os comandos que foram executados no terminal.
 - A maioria dos comandos do Linux tem sinalizadores adicionais que você pode especificar após o comando para fornecer informações adicionais/menos.
 - Vamos usar os comandos *calendar* e *date* novamente, mas desta vez adicionaremos alguns sinalizadores extras:

cal 09 1993
date +%D

```
[mbanze@strge scratch]$ cal 09 1993
    September 1993
Su Mo Tu We Th Fr Sa
                1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30

[mbanze@strge scratch]$ date +%D
07/07/22
[mbanze@strge scratch]$
```

- O comando calendário agora fornece informações para o 9º mês de 1993 (Setembro) e *date* fornece a data no formato mm/dd/aa.
- Existem vários outros sinalizadores que podem ser usados para *cal* e *date*.
- Eles podem ser visualizados usando as páginas de manual (man pages) para estes comandos:

man cal

```
mbanze@strge:~/scratch
CAL(1) User Commands
NAME
    cal - display a calendar
SYNOPSIS
    cal [options] [[[day] month] year]
DESCRIPTION
    cal displays a simple calendar.  If no arguments are specified, the current month is displayed.
OPTIONS
    -l, --one
        Display single month output.  (This is the default.)
    -3, --three
        Display prev/current/next month output.
    -s, --sunday
        Display Sunday as the first day of the week.
    -m, --monday
        Display Monday as the first day of the week.
    -j, --julian
        Display Julian dates (days one-based, numbered from January 1).
    -y, --year
        Display a calendar for the current year.
    -V, --version
        Display version information and exit.
    -h, --help
        Display help screen and exit.
PARAMETERS
    A single parameter specifies the year (1 - 9999) to be displayed; note the year must be fully s
    Two parameters denote the month (1 - 12) and year.
    Three parameters denote the day (1-31), month and year, and the day will be highlighted if
    specified, the current month's calendar is displayed.
Manual page cal(1) line 1 (press h for help or q to quit)
```

A *man page* para `cal` fornece uma lista de sinalizadores adicionais que podem ser usados. Você deve ser capaz de percorrer este documento usando as setas do teclado e pode-se sair de uma página de manual usando `q` e você terminará de volta ao terminal onde poderá digitar comandos novamente.

6.2. Logging Out

- Para sair do shell, use o comando “exit”
- Pressionar “Ctrl+D” no prompt do shell também encerrará o shell
- Saindo de todos os programas deve lhe desconectar.
 - Se estiver em um ambiente de shell único somente texto, saindo do shell deve ser suficiente
 - Em um ambiente de janela, o gerenciador de janelas deve ter um comando de logout para este propósito
- Após o logout, um novo prompt de login deve ser exibido.

6.3. Sintaxe do Comando

- A maioria dos comandos aceita parâmetros.
 - Alguns comandos os exigem;
 - Os parâmetros também são conhecidos como argumentos;

- Por exemplo, echo simplesmente exibe seus argumentos:

```
[mbanze@strge ~]$ echo
[mbanze@strge ~]$ echo The MoRENet_HPC team
The MoRENet_HPC team
[mbanze@strge ~]$
```

- Os comandos diferenciam maiúsculas de minúsculas
 - Geralmente letras minúsculas

```
[mbanze@strge ~]$ echo INAGE
INAGE
[mbanze@strge ~]$ ECHO INAGE
-bash: ECHO: command not found
[mbanze@strge ~]$
```

- Frequentemente você desejará determinar seu directório de trabalho actual e listar os itens no directório actual.
 - Isso pode ser feito usando: pwd e ls

```
[mbanze@strge earthsc]$ pwd
/apps/morenethpc/earthsc
[mbanze@strge earthsc]$ ls
anaconda  CROCO          GEOG      include  NCL        OpenBLAS    python-3.9.9  R-4.1.2          WRFCHM-4.3.1-pnc-gnu8-mpi3
atlas     Delft3DFM_tag68819_bin_intel  grads-2.2.1  lib      nwp4inam   python-3.10.3  R          scilab-6.1.1
bin       flex-2.6.4     IDL        mathworks octave-6.4.0 python-3.10.4  R-4.1.0     WRF-4.3.1-pnc-gnu8-mpi3
[mbanze@strge earthsc]$
```

- Quando você deseja determinar a quantidade de espaço que você tem em seu sistema, você pode usar: df

```
[mbanze@strge earthsc]$ df
Filesystem            1K-blocks      Used    Available Use% Mounted on
devtmpfs              16328444         0    16328444   0% /dev
tmpfs                 16340492         0    16340492   0% /dev/shm
tmpfs                 16340492    108868    16231624   1% /run
tmpfs                 16340492         0    16340492   0% /sys/fs/cgroup
/dev/mapper/centos-root 681667584    29874224    651793360   5% /
/dev/sda2              1038336        281548        756788  28% /boot
/dev/sdb1              961102880    249427480    662830908  28% /apps
/dev/sda1              204580         11476        193104    6% /boot/efi
/dev/sdb2             10652622752    527227924    9588507532   6% /mnt/strage
/dev/mapper/centos-home 276688896     37916332    238772564  14% /home
tmpfs                  3268100         0        3268100   0% /run/user/10004
tmpfs                  3268100         0        3268100   0% /run/user/10005
tmpfs                  3268100         0        3268100   0% /run/user/1000
tmpfs                  3268100         0        3268100   0% /run/user/1004
[mbanze@strge earthsc]$
```

- Há também um comando que pode ser usado para verificar a memória disponível em um sistema: free

```
[mbanze@strge earthsc]$ free
              total        used        free      shared  buff/cache   available
Mem:          32680988    1510272    15106280     104960     16064436     30633268
Swap:         16449532      177152     16272380
```

OpenHPC MoRENet Edição 2022

Tabela 2: Resumo de comandos a serem Usados

Comandos	Significados
touch	Used to create an empty file or update the timestamp of a file
ls	List items in current directory
cat	Concatenate a file/s
echo	Print text to screen
mv	Used to move/rename a file/directory
cp	Used to make a copy of a file/directory
rm	Used to remove a file/directory
pwd	Present working directory
mkdir	Used to make a new directory
cd	Change directory

- Vamos criar um arquivo chamado MoRENet que não contém nenhuma informação e então verificar se o existe listando os itens em nosso directório actual: *touch MoRENet*

```
[mbanze@strge HPC_TRAINING]$ touch MoRENet
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ ls
MoRENet
[mbanze@strge HPC_TRAINING]$
```

- Podemos exibir as informações contidas no arquivo chamado MoRENet: *cat MoRENet*

```
[mbanze@strge HPC_TRAINING]$ cat MoRENet
[mbanze@strge HPC_TRAINING]$
```

Como o arquivo não contém informações dentro dele, não encontramos nada sendo exibido.

Agora vamos colocar algumas informações no arquivo chamado *MoRENet*.

Existem várias maneiras de adicionar informações a um arquivo, mas por enquanto a aula nós usaremos a seguinte abordagem: *echo 'Hello, this is HPC trainig' > MoRENet* e por fim fazer *cat MoRENet*.

```
[mbanze@strge HPC_TRAINING]$ echo 'Hello, this is HPC Training' > MoRENet
[mbanze@strge HPC_TRAINING]$ cat MoRENet
Hello, this is HPC Training
[mbanze@strge HPC_TRAINING]$
```

- Como vimos antes, o echo nos permite imprimir informações na tela, mas ao invés de imprimir as informações na tela estamos redireccionando (>) isso para o arquivo chamado *MoRENet*.
- Em seguida, exibimos o conteúdo do arquivo *MoRENet* (*cat MoRENet*) e descobrimos que o texto da instrução echo está contido nele.

- Agora que nosso arquivo contém algumas informações, não faz sentido tê-lo nomeado *MoRENet*, então o arquivo precisa ser renomeado: `mv MoRENet myfile.txt`

```
[mbanze@strge HPC_TRAINING]$ cat MoRENet
Hello, this is HPC Training
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ ls
MoRENet
[mbanze@strge HPC_TRAINING]$ mv MoRENet myfile.txt
[mbanze@strge HPC_TRAINING]$ ls
myfile.txt
[mbanze@strge HPC_TRAINING]$ cat myfile.txt
Hello, this is HPC Training
[mbanze@strge HPC_TRAINING]$
```

- `mv` nos permite mover um arquivo de nome *MoRENet* para *myfile.txt*.
- Ao listar (`ls`) os itens na pasta actual descobrimos que não existe mais um arquivo chamado *MoRENet*, mas existe um arquivo chamado *myfile.txt*.

Exibindo o conteúdo de *myfile.txt* (`cat myfile.txt`) vemos que a informação é exactamente a mesma que estava contido no arquivo *MoRENet*

Veja que decidimos adicionar uma extensão `.txt` ao novo nome do arquivo e, isso é apenas para que saibamos que este arquivo contém texto.

Dar extensões de arquivos permite que você saiba o que está dentro desses arquivos.

- Se precisarmos fazer uma cópia do nosso arquivo:

```
cp myfile.txt myfile2.txt
```

```
[mbanze@strge HPC_TRAINING]$ ls
myfile.txt
[mbanze@strge HPC_TRAINING]$ cp myfile.txt myfile2.txt
[mbanze@strge HPC_TRAINING]$ ls
myfile2.txt  myfile.txt
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ cat myfile.txt
Hello, this is HPC Training
[mbanze@strge HPC_TRAINING]$ cat myfile2.txt
Hello, this is HPC Training
[mbanze@strge HPC_TRAINING]$
```

- Agora que fizemos uma cópia de *myfile.txt* podemos remover o arquivo original:

- Para excluir um arquivo, use o comando rm ('remove')
- Basta passar o nome do arquivo a ser excluído como argumento

```
[mbanze@strge HPC_TRAINING]$ rm myfile.txt
[mbanze@strge HPC_TRAINING]$ ls
myfile2.txt
[mbanze@strge HPC_TRAINING]$
```

- O arquivo e seu conteúdo são removidos
 - Não há *recycle bin* (lixeira)
 - Não há comando "unrm"
- O comando ls pode ser usado para confirmar a exclusão.

- Vamos tentar adicionar uma linha adicional de informações em myfile2.txt:

```
echo 'There are Vitalina, Banze, Nhadelo and Chaimite' > myfile2.txt
```

```
cat myfile2.txt
```

```
[mbanze@strge HPC_TRAINING]$ echo "There're Vitalina, Banze, Nhadelo and Chaimite" > myfile2.txt

[mbanze@strge HPC_TRAINING]$ cat myfile2.txt
There're Vitalina, Banze, Nhadelo and Chaimite
[mbanze@strge HPC_TRAINING]$
```

No final do processo acima myfile2.txt tem apenas o texto **Thre're Vitalina, Banze, Nhadelo and Chaimite**.

O texto **Hello, this is HPC training** parece ter sido removido.

Em vez de adicionar (anexar) ao arquivo myfile2.txt, parece que sobrescrevemos a informação.

Isso ocorre porque > é um sinal usado para escrever e substituir informações, enquanto >> é usado para anexar (append) a um arquivo.

Observe que uma vez que um arquivo é removido no Linux, ele não pode ser recuperado, portanto, tenha muito cuidado antes de remover qualquer arquivos.

- Agora vamos tentar obter myfile2.txt com as duas linhas que gostaríamos de ver:


```
[mbanze@strge HPC_TRAINING]$ ls
myfile2.txt
[mbanze@strge HPC_TRAINING]$ echo 'Hello, this is HPC training'>myfile2.txt
[mbanze@strge HPC_TRAINING]$ cat myfile2.txt
Hello, this is HPC training
[mbanze@strge HPC_TRAINING]$ echo "There're Vitalina, Banze, Nhadelo and Berny">>myfile2.txt
[mbanze@strge HPC_TRAINING]$ cat myfile2.txt
Hello, this is HPC training
There're Vitalina, Banze, Nhadelo and Berny
[mbanze@strge HPC_TRAINING]$
```

- Conforme dissemos antes, há muitas maneiras de criar um arquivo
- Um dos mais simples é com o comando cat:

```
[mbanze@strge scratch]$ cat > HPC_list
Enga. Vitalina
Dr. Chaimite
Dr. Banze
Dr. Nhadelo

[mbanze@strge scratch]$
```

- Observe o sinal de maior que (>) - isso é necessário para criar o arquivo
- O texto digitado é gravado em um arquivo com o nome especificado
- Pressione Ctrl+D após uma quebra de linha para indicar o final do arquivo
 - O próximo prompt do shell é exibido
- ls demonstra a existência do novo arquivo

```
[mbanze@strge scratch]$ ls
2m temp.png ECMWF ERA-40_subset.nc HPC_list index.html job.169.log job.198.log martilio.py my_first_file.hdf5 tcc.py teste.r testscript.slm
[mbanze@strge scratch]$
[mbanze@strge scratch]$ cat HPC_list
Enga. Vitalina
Dr. Chaimite
Dr. Banze
Dr. Nhadelo

[mbanze@strge scratch]$
```

- Observe que nenhum sinal de maior que é usado
- O texto no arquivo é exibido imediatamente:
 - Iniciando na linha após o comando
 - Antes do próximo prompt do Shell

6.4. Directórios

- Até este ponto trabalhamos apenas com arquivos, mas agora vamos olhar para directórios (também conhecidos como pastas).
- Começamos criando um directório:

```
mkdir folder1
```

```
ls
```

```
[mbanze@strge HPC_TRAINING]$ ls
myfile2.txt
[mbanze@strge HPC_TRAINING]$ mkdir folder1
[mbanze@strge HPC_TRAINING]$ ls
folder1  myfile2.txt
[mbanze@strge HPC_TRAINING]$
```

- Note que os arquivos e pastas são coloridos de forma diferente, com o caso acima mostrando arquivos em branco e **pastas** em azul.

Importa dizer que as cores dos arquivos e pastas no Windows, Linux e MacOS podem diferir das fornecidas, mas todos os sistemas Linux sempre terão arquivos e pastas em cores diferentes.

- Em seguida, vamos dar uma olhada em como mover para uma pasta:

```
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ pwd
/home/mbanze/HPC_TRAINING
[mbanze@strge HPC_TRAINING]$ ls
folder1  myfile2.txt
[mbanze@strge HPC_TRAINING]$ cd folder1/
[mbanze@strge folder1]$ pwd
/home/mbanze/HPC_TRAINING/folder1
[mbanze@strge folder1]$ ls
[mbanze@strge folder1]$
```

- Primeiro, verificamos nosso directório de trabalho actual (pwd), depois mudamos o directório para nossa nova pasta (cd folder1) e, finalmente, damos uma olhada em nosso novo directório de trabalho actual (pwd) e nota que nossa pasta muda de:
 - **/home/mbanze/HPC_TRAINING/folder1**

Quando fizemos ls, não apareceu nada no folder1, mas o que aconteceu com o arquivo myfile2.txt que estava presente quando criamos a folder1?

Esses arquivos ainda estão lá, mas não estão na folder1, em vez disso, estão localizados um directório de volta:

```
cd ..
```

```
[mbanze@strge folder1]$ ls
[mbanze@strge folder1]$ pwd
/home/mbanze/HPC_TRAINING/folder1
[mbanze@strge folder1]$ cd ..
[mbanze@strge HPC_TRAINING]$ pwd
/home/mbanze/HPC_TRAINING
[mbanze@strge HPC_TRAINING]$ ls
folder1  myfile2.txt
[mbanze@strge HPC_TRAINING]$
```

- Para obter um directório de volta, precisamos primeiro alterar o directório (cd ..), onde o .. diz Linux para voltar um directório.
- Em seguida, acabamos de volta de onde começamos quando criamos folder1.
- Agora você vê que o arquivo Linux myfile2.txt ainda está presente.

- Crie um novo directório chamado folder2 e mova-o para folder1:

```
[mbanze@strge HPC_TRAINING]$ mkdir folder2
[mbanze@strge HPC_TRAINING]$ ls
folder1  folder2  myfile2.txt
[mbanze@strge HPC_TRAINING]$ mv folder2 folder1
[mbanze@strge HPC_TRAINING]$ ls
folder1  myfile2.txt
[mbanze@strge HPC_TRAINING]$ pwd
/home/mbanze/HPC_TRAINING
[mbanze@strge HPC_TRAINING]$ cd folder1/
[mbanze@strge folder1]$ pwd
/home/mbanze/HPC_TRAINING/folder1
[mbanze@strge folder1]$ ls
folder2
[mbanze@strge folder1]$
```

- Algumas sintaxes importantes a serem observadas ao alterar os directórios:
 - cd . (Significa ficar no directório actual);
 - cd .. (significa voltar um directório);
 - cd ../../ (significa voltar dois directórios).

Agora queremos retornar ao directório de onde começamos originalmente e estar em folder2 ele nos coloca dois directórios dessa posição:

```
[mbanze@strge folder1]$
[mbanze@strge folder1]$ cd folder2/
[mbanze@strge folder2]$ ls
[mbanze@strge folder2]$ cd ../../
[mbanze@strge HPC_TRAINING]$ pwd
/home/mbanze/HPC_TRAINING
[mbanze@strge HPC_TRAINING]$ ls
folder1  myfile2.txt
[mbanze@strge HPC_TRAINING]$
```

Copie myfile2.txt para a folder1 e verifique se o conteúdo do arquivo ainda é o mesmo:

```
[mbanze@strge HPC_TRAINING]$ ls
folder1  myfile2.txt
[mbanze@strge HPC_TRAINING]$ cp myfile2.txt folder1/
[mbanze@strge HPC_TRAINING]$ ls
folder1  myfile2.txt
[mbanze@strge HPC_TRAINING]$ cd folder1/
[mbanze@strge folder1]$ pwd
/home/mbanze/HPC_TRAINING/folder1
[mbanze@strge folder1]$ ls
folder2  myfile2.txt
[mbanze@strge folder1]$ cat myfile2.txt
Hello, this is HPC training
There're Vitalina, Banze, Nhadelo and Berny
[mbanze@strge folder1]$
```

Por fim, removeremos folder1 e tudo o que ele contém (myfile2.txt e folder2):

```
[mbanze@strge folder1]$ ls
folder2  myfile2.txt
[mbanze@strge folder1]$ cd ..
[mbanze@strge HPC_TRAINING]$ ls
folder1  myfile2.txt
[mbanze@strge HPC_TRAINING]$ rm folder1/
rm: cannot remove 'folder1/': Is a directory
[mbanze@strge HPC_TRAINING]$
```

Como já estamos dentro do folder1, primeiro precisamos mover um directório de volta (cd ..).

Quando tentamos remover a pasta, ocorre um erro informando que a folder1 é um directório.

Enquanto o rm funciona sozinho para remoção de arquivos, precisamos usar um sinalizador adicional quando lidando com directórios:

```
[mbanze@strge HPC_TRAINING]$ rm -rv folder1/
removed directory: 'folder1/folder2'
removed 'folder1/myfile2.txt'
removed directory: 'folder1/'
[mbanze@strge HPC_TRAINING]$ ls
myfile2.txt
[mbanze@strge HPC_TRAINING]$
```

Usamos o sinalizador -r ao remover directórios para dizer ao Linux para remover os arquivos recursivamente.

Eu incluí a opção -v adicional, que significa verbose, para exibir a remoção processo como acontece.

Você notará que a primeira coisa que é removida é folder2, seguida por myfile2.txt e finalmente folder1.

Com base no comando list no final da imagem anterior vemos que ainda existe um arquivo chamado myfile2.txt que existe em nosso directório pessoal, então vamos removê-lo também, já que não precisamos mais:

```
[mbanze@strge HPC_TRAINING]$ ls
myfile2.txt
[mbanze@strge HPC_TRAINING]$ rm myfile2.txt
[mbanze@strge HPC_TRAINING]$
```

Tabela 3: Resumo de comandos a serem usados

Command	Significado
nano	Editor de texto de linha de comando no Linux
Ctrl+X>Y>Enter	Sequência usada ao salvar alterações em um arquivo criado/editado em nano
Ls	Listar itens no directório actual
Cat	Concatenar um arquivo/s
Rm	Usado para remover um arquivo/directório

- Como nas aulas anteriores, todos os comandos fornecidos a partir daqui em diante são comandos que podem ser digitados/executados directamente em um terminal.
- Na aula anterior criamos um arquivo texto usando echo e depois usamos a instrução echo para um arquivo.
- Existem vários editores de texto disponíveis no Linux que permitem criar e editar arquivos.
- Estes incluem editores de linha de comando como:
 - [nano](#)
 - [vim](#)
 - [emacs](#)
 - [The Nice Editor](#)
 - [tilde](#)
- Existem muitos outros editores Linux disponíveis e as listas fornecidas acima são meramente alguns dos editores mais populares que são usados.
- O editor que você escolhe usar é totalmente com você, mas para os propósitos deste curso
- Vou usar o nano.
- Vamos criar um arquivo chamado hpc.txt usando nano:

nano hpc.txt

```
mbanze@strge:~/HPC_TRAINING
GNU nano 2.3.1 File: hpc.txt

[ New File ]

Get Help WriteOut Read File [ New File ] Prev Page Cut Text Cur Pos
Exit Justify Where Is Next Page UnOut Text To Spell
```

- Note que não vemos mais o ~\$ no início da linha.
- Isso nos diz que não estamos mais no terminal e agora estamos dentro de um editor de texto.
- Quaisquer comandos digitados aqui serão colocados directamente no arquivo que chamamos hpc.txt.

Agora vamos colocar algum texto em nosso novo arquivo:

- 1) HPC é a capacidade de processar dados e realizar cálculos complexos em altas velocidades.
- 2) A MoRENet possui dois equipamentos de HPC

```
GNU nano 2.3.1 File: hpc.txt

HPC é a capacidade de processar dados e realizar cálculos complexos em altas velocidades.
A MoRENet possui dois equipamentos de HPC.

^G Get Help      ^O WriteOut      ^R Read File     ^V Prev Page     ^X Cut Text      ^C Cur Pos
^X Exit          ^U Justify       ^W Where Is      ^W Next Page     ^U UnCut Text    ^T To Spell
```

- Agora que temos informações em hpc.txt, precisamos salvar as alterações.
- Na parte inferior da tela há várias opções diferentes, como Obter Ajuda, Sair, etc.
- O ^ antes de cada letra refere-se à tecla Ctrl do seu teclado.
- O que queremos fazer é Sair, então precisamos usar Ctrl+X:
- Em seguida, nos perguntará se desejamos salvar o buffer modificado (o que significa que queremos que salve nossas alterações), então apertamos Y.
- Finalmente, somos questionados sobre o nome do arquivo a ser escrito. Neste caso, desejamos manter o mesmo nome, então tudo o que fazemos aqui é pressionar tecla Enter.
- Estamos agora de volta ao terminal, pois podemos ver nosso comando anterior, bem como o ~\$.
- Vamos dar uma olhada se nosso arquivo foi criado e se ele tem as informações que nós digitado:

ls

cat hpc.txt

```
[mbanze@strge HPC_TRAINING]$ nano hpc.txt
[mbanze@strge HPC_TRAINING]$ ls
hpc.txt
[mbanze@strge HPC_TRAINING]$ cat hpc.txt

HPC é a capacidade de processar dados e realizar cálculos complexos em altas velocidades.
A MoRENet possui dois equipamentos de HPC.
[mbanze@strge HPC_TRAINING]$
```

- Listar os itens no directório actual (ls) mostra que temos um arquivo chamado hpc.txt presente.

- A verificação do conteúdo do arquivo (cat hpc.txt) revela que o arquivo possui as informações que inserimos no nano.
- Fazer uso de editores de texto como o nano são consideravelmente importantes ao escrever bash scripts, pois muitas vezes você pode acabar com muitas linhas de código, o que seria complicado para colocar em um arquivo com algo como echo.
- Remova o arquivo hpc.txt e confirme se ele não existe mais.
- Isso nos leva ao fim dos Editores de Texto.

NÃO haverá exercício para esta seção, pois faremos uso extensivo de texto editores quando chegarmos à parte de scripts bash deste curso.

- Agora vamos criar vários arquivos chamados test1.txt, test2.txt, test3.txt, test4.csv e test5.csv:

```
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ touch test1.txt test2.txt test3.txt test4.csv test5.csv
[mbanze@strge HPC_TRAINING]$ ls
hpc.txt test1.txt test2.txt test3.txt test4.csv test5.csv
[mbanze@strge HPC_TRAINING]$
```

- Note que é possível criar vários arquivos usando um único comando de touch e em vez de ter um comando separado para a criação de cada arquivo.
- Digamos que desejamos remover os arquivos que terminam com .csv:

```
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ ls
hpc.txt test1.txt test2.txt test3.txt test4.csv test5.csv
[mbanze@strge HPC_TRAINING]$ rm test4.csv test5.csv
[mbanze@strge HPC_TRAINING]$ ls
hpc.txt test1.txt test2.txt test3.txt
[mbanze@strge HPC_TRAINING]$ touch test4.csv test5.csv
[mbanze@strge HPC_TRAINING]$ ls
hpc.txt test1.txt test2.txt test3.txt test4.csv test5.csv
[mbanze@strge HPC_TRAINING]$ rm *.csv
[mbanze@strge HPC_TRAINING]$ ls
hpc.txt test1.txt test2.txt test3.txt
[mbanze@strge HPC_TRAINING]$
```

- Uma maneira de remover os arquivos seria digitar os nomes de cada arquivo após o comando rm.
- Desde que removi os arquivos test4.csv e test5.csv, preciso recriá-los usando o touch.
- E se tivermos 100 ou mesmo 1000 de arquivos .csv, levará um tempo considerável para digitar cada nome, então para economizar tempo podemos usar o caractere *. Isso é usado como um espaço reservado e representa qualquer texto/caractere que possa aparecer antes da extensão .csv.
- Podemos remover arquivos de outra forma. Por exemplo, vamos remover test2.txt:


```
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ ls
hpc.txt  test1.txt  test2.txt  test3.txt
[mbanze@strge HPC_TRAINING]$ rm *2*
[mbanze@strge HPC_TRAINING]$ ls
hpc.txt  test1.txt  test3.txt
[mbanze@strge HPC_TRAINING]$
```

- O comando remove acima removerá todos os arquivos que contêm o número 2, independentemente do que vem antes ou depois do número. Nesse caso, é apenas test2.txt que atende ao critério e, portanto, é removido.
- Por fim, queremos remover todos os arquivos que começam com test e terminam com .txt, mas não nos preocupamos com o número que aparece entre eles:

```
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ ls
shpc.txt  test1.txt  test3.txt
[mbanze@strge HPC_TRAINING]$ rm test*.txt
[mbanze@strge HPC_TRAINING]$ ls
hpc.txt
[mbanze@strge HPC_TRAINING]$
```

- O comando acima remove os arquivos restantes (test1.txt e test3.txt).

6.5. Flag adicionais e localização de arquivos

Tabela 4: Resumo de Comandos a serem usados

Comando	Significado
echo	Print text to screen
ls	List items in current directory
mv	Used to move/rename a file/directory
find	Command used to find specific files in Linux
cd	Change directory
mkdir	Used to make a new directory
rm	Used to remove a file/directory

Crie um arquivo chamado *myinfo.txt* que contenha algum texto:

```
echo 'HPC is powerful' > myinfo.txt
```

```
ls
```

```
[mbanze@strge HPC_TRAINING]$ echo 'HPC is powerful' >myinfo.txt
[mbanze@strge HPC_TRAINING]$ ls
myinfo.txt
[mbanze@strge HPC_TRAINING]$
```

No meu caso ls mostra que tenho os arquivo myinfo.txt contidos no meu directório actual.

- Se eu quiser obter mais informações sobre meus arquivos, preciso adicionar um sinalizador adicional ao ls: `ls -l`

```
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ ls -l
total 4
-rw-rw-r-- 1 mbanze mbanze 16 Jul 12 08:57 myinfo.txt
[mbanze@strge HPC_TRAINING]$
```

- As informações fornecidas incluem as permissões do arquivo/directório (quem tem permissão para acessar e fazer alterações no conteúdo), o usuário que criou o arquivo, a qual grupo o usuário pertence, o tamanho do arquivo e a data e hora em que o arquivo/directório foi criado ou alterado.
- Crie outro arquivo chamado `yourinfo.txt` que contenha algum texto: `echo 'Com o HPC não iremos levar meses ou muito tempo a processarmos dados' > yourinfo.txt`

`ls -l`

```
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ echo 'Com o HPC não iremos levar meses ou muito tem
po a processarmos dados' > yourinfo.txt
[mbanze@strge HPC_TRAINING]$ ls -l
total 8
-rw-rw-r-- 1 mbanze mbanze 16 Jul 12 08:57 myinfo.txt
-rw-rw-r-- 1 mbanze mbanze 70 Jul 12 09:32 yourinfo.txt
[mbanze@strge HPC_TRAINING]$
```

- Nosso directório agora tem dois arquivos que foram criados em momentos diferentes e têm tamanhos diferentes.
- Crie um directório chamado `linux` e mova os arquivos `txt` para este directório:

```
mkdir linux
mv *.txt temp/
ls -l
```

```
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ mkdir linux
[mbanze@strge HPC_TRAINING]$ mv *.txt linux/
[mbanze@strge HPC_TRAINING]$ ls -l
total 0
drwxrwxr-x 2 mbanze mbanze 44 Jul 12 09:43 linux
[mbanze@strge HPC_TRAINING]$
```

- No caso actual, sabemos que nossos dois arquivos txt estão contidos no directório linux, pois acabamos de movê-los para lá.
- No entanto, e se nós criamos esses arquivos 6 meses ou 2 anos atrás e os colocamos em um directório com 20 subdirectórios, cada um com seu próprio conjunto de arquivos e temos um total de 1 milhão de arquivos que precisamos pesquisar para encontrar os arquivos que estamos procurando, não faz sentido ir para cada directório/subdirectório um de cada vez antes de localizar o que estamos procurando.
- Para o cenário acima, precisamos fazer uso de:

```
find . -name '*.txt'
```

```
[mbanze@strge HPC_TRAINING]$ find . -iname '*.txt'
./linux/myinfo.txt
./linux/yourinfo.txt
[mbanze@strge HPC_TRAINING]$
```

```
[mbanze@strge HPC_TRAINING]$ find . -name '*info.txt'
./linux/myinfo.txt
./linux/yourinfo.txt
[mbanze@strge HPC_TRAINING]$
```

- O comando *find* diz ao Linux que ele deve começar a procurar todos os arquivos do directório actual (.), para cima (ou seja, directórios e subdirectórios que existem em nosso directório de trabalho actual) e fornecer uma lista de arquivos que termina com .txt.
- Renomeie um dos ficheiros execute novamente o comando *find*:

```
cd linux
mv myinfo.txt myinfo.TXT
cd ..
```

```
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ cd linux/
[mbanze@strge linux]$ mv myinfo.txt myinfo.TXT
[mbanze@strge linux]$ ls
myinfo.TXT  yourinfo.txt
[mbanze@strge linux]$ cd ..
[mbanze@strge HPC_TRAINING]$ pwd
/home/mbanze/HPC_TRAINING
[mbanze@strge HPC_TRAINING]$ find . -name '*info.txt'
./linux/yourinfo.txt
[mbanze@strge HPC_TRAINING]$
```

- A entrada myinfo.TXT não aparece em nossa lista.
- Isso ocorre porque *find*, como a maioria dos comandos do Linux, diferencia maiúsculas de minúsculas.
- Para remover a distinção entre maiúsculas e minúsculas, podemos fazer o seguinte:

```
find . -iname '*info.txt'
```

```
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ find . -iname '*info.txt'
./linux/yourinfo.txt
./linux/myinfo.TXT
[mbanze@strge HPC_TRAINING]$
```

- Agora vemos que a entrada myinfo.TXT está presente.
- Ao usar -iname em vez de -name, estamos dizendo ao find para procurar arquivos que contenham o texto fornecido entre aspas, independentemente de os arquivos encontrados estarem em letras maiúsculas ou minúsculas.
- Remova a pasta linux e todo o seu conteúdo:

```
rm -rv linux
ls
```

```
[mbanze@strge HPC_TRAINING]$ ls
linux
[mbanze@strge HPC_TRAINING]$ cd linux/
[mbanze@strge linux]$ ls
myinfo.TXT  yourinfo.txt
[mbanze@strge linux]$ cd ..
[mbanze@strge HPC_TRAINING]$ rm -rv linux/
removed 'linux/yourinfo.txt'
removed 'linux/myinfo.TXT'
removed directory: 'linux/'
[mbanze@strge HPC_TRAINING]$ ls
[mbanze@strge HPC_TRAINING]$
```

OpenHPC MoRENet E

Tabela 5: Resumo de comandos a serem usados

Comando	Significado
nano	Command line text editor in Linux
Ctrl+X > Y > Enter	Sequence used when saving changes to a file created/edited in nano
cat	Concatenate a file/s
ls	List items in current directory
grep	Used to find text in a file
sed	Used to find and replace text in a file

- Criemos um arquivo chamado hpc_info.txt usando nano com a seguinte informação de dois parágrafos:

Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster;

Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gb.

```
[mbanze@strge HPC_TRAINING]$ nano hpc_info.txt
[mbanze@strge HPC_TRAINING]$ ls
hpc_info.txt
[mbanze@strge HPC_TRAINING]$ cat hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster;
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gb.
[mbanze@strge HPC_TRAINING]$
```

- Para pesquisar uma palavra em um arquivo, usamos:

grep 'cluster' hpc_info.txt

```
[mbanze@strge HPC_TRAINING]$ grep -n 'cluster' hpc_info.txt
1:Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster;
2:Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gb.
[mbanze@strge HPC_TRAINING]$
```

- O grep pesquisará o arquivo hpc_info.txt e procurará as palavras/caracteres que correspondem ao que é fornecido nas aspas e destacar.
- Agora você encontra os números de linha fornecidos à esquerda correspondentes à linha onde a palavra/caractere aparece dentro do arquivo hpc_info.txt.

- Pesquise no arquivo hpc_info.txt as informações da palavra:

```
grep '100 GB' hpc_info.txt
```

```
[mbanze@strge HPC_TRAINING]$ grep '100 GB' hpc_info.txt
[mbanze@strge HPC_TRAINING]$
```

- Isso não parece produzir nada, apesar do arquivo que contém a palavra.
- Como na maioria dos comandos do Linux, o grep diferencia maiúsculas de minúsculas e a palavra no arquivo contém um “b” minúsculo.
- Existem duas abordagens que podem ser usadas para obter o resultado que desejamos:

```
grep '100 Gb' randominfo.txt
grep -i '100 GB' randominfo.txt
```

```
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$ grep -i '100 GB' hpc_info.txt
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gb.
[mbanze@strge HPC_TRAINING]$ grep -i '100 Gb' hpc_info.txt
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gb.
[mbanze@strge HPC_TRAINING]$
```

- Agora precisamos corrigir os erros contidos no arquivo.
- Começaremos alterando a palavra Gb:

```
sed -e 's/Gb/Gbyte/g' hpc_info.txt
cat hpc_info.txt
```

```
[mbanze@strge HPC_TRAINING]$ sed -e 's/Gb/Gbyte/g' hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster;
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gbyte.

[mbanze@strge HPC_TRAINING]$ cat hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster;
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gb.

[mbanze@strge HPC_TRAINING]$
```

- sed é o comando usado quando você deseja localizar e substituir texto.
- O -e diz ao sed para fazer as alterações e exibir o resultado na tela. Como você pode ver na imagem acima, o resultado de como será o novo texto é exibido na tela, porém nenhuma alteração ocorre no arquivo hpc_info.txt. A exibição do conteúdo do arquivo (cat hpc_info.txt) mostra que as informações originais permanecem.
- O 's/Gb/Gbyte/g' diz ao sed para examinar cada linha de texto (s) para encontrar a palavra Gb e substituí-la por Gbyte e fazer isso para cada instância em que a palavra aparecer (g).
- O comando sed acima não é prático pois se existisse uma palavra correcta iria adicionar yte e ficando Gbyteyte (teste no seu computador) fornecendo o resultado indesejado.

```
[mbanze@strge HPC_TRAINING]$ sed -e 's/Gb/GB/g' hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster;
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 GB.

[mbanze@strge HPC_TRAINING]$ echo "Eu tenho 1000 Gbyte para processar." >>hpc_info.txt
[mbanze@strge HPC_TRAINING]$ cat hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster;
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gb.

Eu tenho 1000 Gbyte para processar.
[mbanze@strge HPC_TRAINING]$ sed -e 's/Gb/Gbyte/g' hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster;
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gbyte.

Eu tenho 1000 Gbyteyte para processar.
[mbanze@strge HPC_TRAINING]$ cat hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster;
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gb.

Eu tenho 1000 Gbyte para processar.
```

- Precisamos fazer algumas alterações no comando:

```
sed -e '2s/Gb/Gbyte/g' hpc_info.txt
sed -i '2s/Gb/Gbyte/g' hpc_info.txt
```

```
[mbanze@strge HPC_TRAINING]$ cat hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gb
Eu tenho 1000 Gbyet para processar
[mbanze@strge HPC_TRAINING]$ sed -e 's/Gb/Gbyet/g' hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gbyet
Eu tenho 1000 Gbyetyet para processar
[mbanze@strge HPC_TRAINING]$ sed -e '2s/Gb/Gbyet/g' hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gbyet
Eu tenho 1000 Gbyet para processar
[mbanze@strge HPC_TRAINING]$ cat hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gb
Eu tenho 1000 Gbyet para processar
[mbanze@strge HPC_TRAINING]$ sed -i '2s/Gb/Gbyet/g' hpc_info.txt
[mbanze@strge HPC_TRAINING]$ cat hpc_info.txt
Os comandos que estou usando agora vão ser uteis quando eu for a usar o cluster
Eu estou ansioso em aceder ao cluster para processar o meu dado de 100 Gbyet
Eu tenho 1000 Gbyet para processar
[mbanze@strge HPC_TRAINING]$
```

6.6. Variáveis e Loops

Tabela 6: Resumo dos comandos usados nesta secção

Command	Meaning
echo	Print text to screen
var=XX	Create a variable called var that stores XX
\$var	Print the contents of the variable var
for var in XX; do ...; done	Perform an iterative task
cat	Concatenate a file/s
history	Provides the terminal command history
	Takes the output of one command and passes it as an input to another
grep	Used to find text in a file

- Variáveis são usadas como espaços reservados que podem armazenar informações que você pode usar mais tarde.
Isso é consideravelmente importante ao usar scripts bash, algo que veremos mais adiante neste curso.
- Criaremos uma variável chamada num que armazena o número 10 e então usaremos echo para imprimir a variável na tela.

```
num=10
```

```
eco $num
```

```
echo 'O número armazenado na variável é $num'
```

```
echo 'O número armazenado na variável é' $num
```

```
[mbanze@strge ~]$ num=10
[mbanze@strge ~]$ echo "O numero armazenado na variavel eh $num"
O numero armazenado na variavel eh 10
[mbanze@strge ~]$ echo 'O numero armazenado na variavel eh $num'
O numero armazenado na variavel eh $num
[mbanze@strge ~]$ echo 'O numero armazenado na variavel eh' $num
O numero armazenado na variavel eh 10
[mbanze@strge ~]$
```

- Uma vez que uma variável é declarada, ela pode ser chamada usando \$ seguido do nome da variável.

Observe que com os comandos do Linux nunca deve haver espaços entre o nome da variável, o sinal de igual e o valor atribuído à variável. Se você incluir um espaço, a variável permanecerá vazia. Além disso, quando você fecha um terminal e abre um novo, a variável não existe mais, então você precisará redeclará-la.

Pode haver várias variáveis declaradas e combinadas:

```
num1=4
num2=8
soma=$num1+'$num2
echo $soma
```

```
[mbanze@strge ~]$ $soma
[mbanze@strge ~]$ num1=4
[mbanze@strge ~]$ num2=8
[mbanze@strge ~]$ soma=$num1+'$num2
[mbanze@strge ~]$ echo $soma
4+8
```

- Temos dois números (4 e 8) armazenados nas variáveis num1 e num2, enquanto a variável soma mostra num1 e num2 separados por um sinal de adição.
- Talvez seja mais benéfico ter a variável soma armazenando a solução matemática para a adição dos dois números (produzindo 12 ao fazer echo \$soma), mas veremos como isso pode ser feito mais adiante neste manual.
- Muitas vezes, há casos em que você gostaria de repetir uma tarefa várias vezes, mas não deseja fazer a repetição manualmente.
- É aqui que o uso de loops pode ser muito benéfico.
- A construção de um bash for loop tem a seguinte aparência:
for variable in something; do task/command; done
- Você precisa ter uma variável (que, como mencionado acima, é um espaço reservado). Depois disso, você executa alguma tarefa na variável e quando essa tarefa é concluída, o loop é concluído.
- Primeiramente, tentaremos imprimir os números de 01 a 10 na tela:

```
echo {01..10}
```

```
[mbanze@strge ~]$ echo {01..10}
01 02 03 04 05 06 07 08 09 10
[mbanze@strge ~]$ echo {1..10}
1 2 3 4 5 6 7 8 9 10
[mbanze@strge ~]$
```

- {01..10} é uma ferramenta de contagem no Linux que fornecerá os números de 01 a 10 em etapas de 1.
- Se isso for impresso na tela, teremos todos os números sendo exibidos em uma única linha.

- Agora, se quisermos ter a mesma coisa impressa como linhas separadas:

```
for i in {01..10};
do
```

```
echo $i;
done
```

```
[mbanze@strge ~]$
[mbanze@strge ~]$ for i in {01..10};
> do
> echo $i;
> done
01
02
03
04
05
06
07
08
09
10
[mbanze@strge ~]$
```

- No caso acima, você notará que existem linhas com um caractere>. NÃO significam (linhas) redireccionamento, mas em vez disso, elas estão mostrando que você está dentro de um loop e quando o loop estiver completo, você será direccionado de volta ao terminal ~\$.
 - A variável usada acima é chamada de i. Na primeira iteração i assume o valor 01 e quando entramos no loop tudo o que fazemos é imprimir este valor na tela (echo \$i).
 - Em seguida, fazemos um loop (ou voltamos) para uma segunda iteração, onde agora eu obtém o valor 02 e isso é impresso na tela.
 - O processo continua até que i obtenha o valor de 10 e isso seja impresso na tela.
 - Depois disso, o loop pára, pois solicitamos apenas que eu vá de 01 a 10.
-
- Vamos dar uma olhada no que acontece se escolhermos não imprimir a variável dentro do loop:

```
for i in {01..10};
do
echo teste;
done
```

```
[mbanze@strge ~]$ for i in {01..10};
> do
> echo teste;
> done
teste
teste
teste
teste
teste
teste
teste
teste
teste
teste
[mbanze@strge ~]$
```

- O loop ainda funciona, mas em vez de imprimir os números de 01 a 10 na tela, tudo o que vemos é a palavra teste impressa 10 vezes.
- Os loops também podem ser usados em conjunto com texto em vez de números:

```
for names in 'banze' 'baptista' 'chaimite' 'virgilio' 'ashley' 'viegas' 'augusto' 'lucia';
do
echo $names;
done
```

Observe que você pode fornecer suas variáveis com o nome que desejar. Eles não precisam ser iguais aos nomes que usados acima.

Apenas lembre-se de que quando você imprime isso na tela, o nome que você fornece depois **for** deve corresponder ao nome usado após o **echo**.

- Agora percorremos os nomes individuais das pessoas e imprimimos cada nome na tela (echo \$names).

Também podemos iterar através de informações contidas em um arquivo:

```
echo 'line1' > somefile.txt
echo 'line2' >> somefile.txt
echo 'line3' >> somefile.txt
cat somefile.txt
for lines in `cat somefile.txt`;
do
echo $lines;
done
```

6.7. Bash scripting Parte 1

Nas aulas anteriores apenas executamos comandos dentro do terminal!

Embora esta seja uma boa abordagem para usar ao executar comandos simples ou se você deseja testar algo, ela não é muito eficiente. Por exemplo, se você declarar uma variável no terminal, no momento em que você sair do terminal e abrir um novo, essa variável desaparecerá.

Geralmente é uma boa ideia colocar seus comandos do Linux em um script, o que permitirá que você use os comandos presentes no script sempre que quiser.

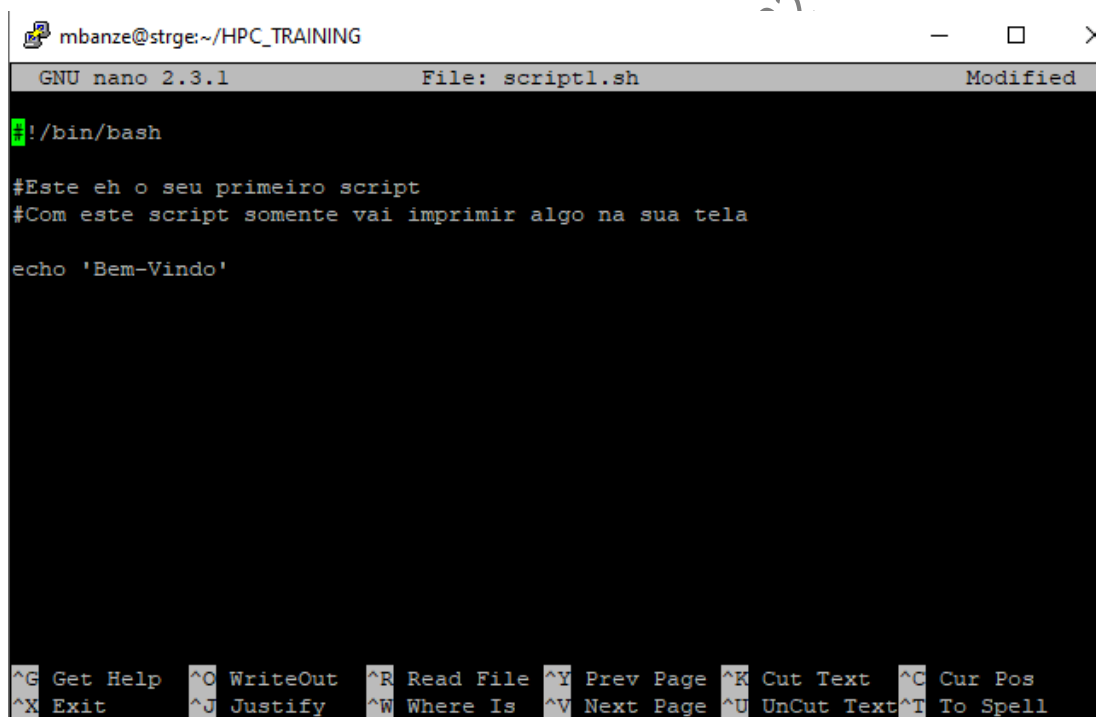
Scripts/Scripting também permite expandir as informações contidas nele, então o que pode ter começado como um script muito simples pode evoluir para algo mais complexo que você pode executar para processar dados de suas simulações, por exemplo.

A escrita de scripts é melhor feita em um editor de texto. Para a parte de script deste curso, usarei o nano como meu editor de texto, mas você pode usar o que achar mais confortável, como vim por exemplo. Primeiro, vamos criar uma pasta onde vamos armazenar todos os scripts que escrevemos a partir deste ponto:

```
mkdir Linux_scripts
cd Linux_scripts
pwd
```

Agora vamos fazer nosso primeiro script chamado script1.sh:

```
nano script1.sh
```



```
mbanze@strge:~/HPC_TRAINING
GNU nano 2.3.1      File: script1.sh      Modified
#!/bin/bash
#Este eh o seu primeiro script
#Com este script somente vai imprimir algo na sua tela
echo 'Bem-Vindo'

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell
```

Use Ctrl+X > Y > Enter para sair do nano, salve as alterações e volte ao terminal.

Verifique se o arquivo existe e se possui as informações que solicitamos:

```
ls
```

```
cat script1.sh
```

Nomeamos nosso arquivo script1.sh. A primeira parte do script poderia ter o nome que você quiser, mas a extensão do arquivo deve ser .sh. Essa extensão é usada para qualquer script bash, assim como qualquer arquivo que termine com .py é usado para scripts escritos para python.

A primeira linha de um script bash é chamada de shebang (`#!/bin/bash`) e aponta para a localização do seu programa/binário que será usado para executar o script.

Todas as linhas que aparecem após o shebang que começam com `#` são usadas para comentários e são ignoradas pelo bash.

É bom ter comentários em seus scripts para permitir que você saiba o que as diferentes partes do script fazem, bem como para ajudar outras pessoas que possam fazer uso de scripts posteriormente.

Também temos uma declaração de `echo` presente em nosso *script* e como sabemos de aulas anteriores, isso deve imprimir algo (“Bem-Vindo”) na tela.

Agora que o script está escrito, vamos tentar executá-lo. A primeira vez que você precisar executar um script bash, verifique se ele é executável:

```
chmod u+x script1.sh
```

```
ls
```

Modificamos o arquivo e o tornamos executável para o usuário (`chmod u+x script1.sh`).

Uma vez que este comando tenha sido executado em um arquivo, ele não precisa ser feito novamente, pois o arquivo agora permanecerá executável.

Você notará que o nome do arquivo muda para verde após ser modificado.

```
[mbanze@strge HPC_TRAINING]$ chmod u+x script1.sh
[mbanze@strge HPC_TRAINING]$ ls -l
total 132
-rwxrwxr-x 1 mbanze mbanze 8424 Jul 20 14:20 a.out
-rw-rw-r-- 1 mbanze mbanze 36970 Jul 20 15:09 funcao_seno.png
-rw-rw-r-- 1 mbanze mbanze 36970 Jul 21 09:26 funcao_seno.png.png
-rw-rw-r-- 1 mbanze mbanze 20 Jul 20 15:40 hello.py
-rw-r--w- 1 mbanze mbanze 193 Jul 21 12:02 hpc_info.txt
-rw-rw-r-- 1 mbanze mbanze 636 Jul 20 14:24 job.401.out
-rw-rw-r-- 1 mbanze mbanze 144 Jul 20 15:09 job.414.log
-rw-rw-r-- 1 mbanze mbanze 155 Jul 20 15:41 job.415.log
-rw-r--r-- 1 mbanze mbanze 376 Jul 20 14:22 job.mpi
-rw-rw-r-- 1 mbanze mbanze 262 Jul 20 15:09 python.py
-rw-rw-r-- 1 mbanze mbanze 86 Jul 21 11:54 randominfo.txtty
-rwxrwxr-- 1 mbanze mbanze 119 Aug 2 10:25 script1.sh
drwxrwxrwx 3 mbanze mbanze 278 Aug 1 18:44 test
-rw-rw-r-- 1 mbanze mbanze 988 Jul 20 15:41 teste.slurm
[mbanze@strge HPC_TRAINING]$
```

6.8. Permissions: Octal Representation

Tabela 7: Permissões em arquivos: leitura, edição e execução

Permissão (In Plain English)	Symbolic	Octal
Read, write and execute	Rwx	7
Read and write	rw-	6
Read and execute	r-x	5
read	r--	4
Write and execute	-wx	3
write	-w-	2
execute	--x	1
No permissions	---	0

Para mudar permissões o comando geralmente toma a seguinte forma:

Chmod MODE file There are two ways to represente the MODE:

- (1) Usando symbolic modes (A letters to indicate the categories and permissions);
- (2) Using numeric modes (An octal (base 8) number that represents the modes).

In order to change the permissions of a file using symbolic permissions, use the command format:

Chmod SYMBOLIC/NUMERIC-MODE FILENAME

chmod 400 file: to protect a file against accidental overwriting.

chmod 500 directory: to protect yourself from accidentally removing, renaming or moving files from this directory.

chmod 600 file: A private file only changeable by user who entered this command.

chmod 644 file: a publicly readable file that can only be changed by the issuing user.

chmod 660 file: Users belonging to your group can change this file, others don't have any access to it at all.

chmod 700 file: Protects a file against any access from other users, while the issuing user still has full access.

chmod 755 directory: For files that should be readable and executable by others, but only changeable by the issuing user.

chmod 775 file: standard file sharing mode for a group.

Com o arquivo sendo executável vamos ver o que acontece quando executamos o script1.sh:

./script1.sh

```
[mbanze@strge HPC_TRAINING]$ ./script1.sh
Bem-Vindo
[mbanze@strge HPC_TRAINING]$
```

A única coisa que aparece é a palavra “Bem-Vindo”.

Nenhuma das informações fornecidas em script1.sh antes da instrução *echo* aparece na tela, pois são comentários que só são visíveis quando se abre o *script* em um editor de texto.

Você pode também escrever scripts que aceitam a entrada do usuário que está executando o script:

nano script2.sh

```
GNU nano 2.3.1      File: script2.sh      Modified

#!/bin/bash

# Este script vai aceitar input do usuario e armazenar como uma variavel

echo 'Escreva o seu nome'

read name

echo 'Ola '$name

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell
```

Quando este script é executado, ele deve solicitar que o usuário digite seu nome. Uma vez que o nome é inserido, ele será armazenado em uma variável chamada nome. Por fim, uma saudação será impressa no nome da pessoa.

Use Ctrl+X > Y > Enter para sair do nano, salvar as alterações e voltar ao terminal.

Verifique se o arquivo existe e se possui as informações que solicitamos:

```
[mbanze@strge HPC_TRAINING]$ nano script2.sh
[mbanze@strge HPC_TRAINING]$ ls
a.out          hello.py      job.414.log   python.py     script2.sh
funcao_seno.png hpc_info.txt job.415.log   randominfo.txtxy teste
funcao_seno.png.png job.401.out  job.mpi      script1.sh    teste.slurm
[mbanze@strge HPC_TRAINING]$ cat script2.sh
#!/bin/bash

# Este script vai aceitar input do usuario e armazenar como uma variavel

echo 'Escreva o seu nome'

read name

echo 'Ola '$name

[mbanze@strge HPC_TRAINING]$
```

Como acabamos de criar este script, precisamos modificá-lo antes de executá-lo:

```
chmod u+x script2.sh
```

```
./script2.sh
```

```
[mbanze@strge HPC_TRAINING]$ ls -l
total 136
-rwxrwxr-x 1 mbanze mbanze 8424 Jul 20 14:20 a.out
-rw-rw-r-- 1 mbanze mbanze 36970 Jul 20 15:09 funcao_seno.png
-rw-rw-r-- 1 mbanze mbanze 36970 Jul 21 09:26 funcao_seno.png.png
-rw-rw-r-- 1 mbanze mbanze 20 Jul 20 15:40 hello.py
-rw-r--w- 1 mbanze mbanze 193 Jul 21 12:02 hpc_info.txt
-rw-rw-r-- 1 mbanze mbanze 636 Jul 20 14:24 job.401.out
-rw-rw-r-- 1 mbanze mbanze 144 Jul 20 15:09 job.414.log
-rw-rw-r-- 1 mbanze mbanze 155 Jul 20 15:41 job.415.log
-rw-r--r-- 1 mbanze mbanze 376 Jul 20 14:22 job.mpi
-rw-rw-r-- 1 mbanze mbanze 262 Jul 20 15:09 python.py
-rw-rw-r-- 1 mbanze mbanze 86 Jul 21 11:54 randominfo.txtxy
-rwxrw-r-- 1 mbanze mbanze 119 Aug 2 10:25 script1.sh
-rw-rw-r-- 1 mbanze mbanze 144 Aug 2 11:24 script2.sh
drwxrwxrwx 3 mbanze mbanze 278 Aug 1 18:44 teste
-rw-rw-r-- 1 mbanze mbanze 988 Jul 20 15:41 teste.slurm
[mbanze@strge HPC_TRAINING]$ chmod +x script2.sh
[mbanze@strge HPC_TRAINING]$ ls -t
script2.sh randominfo.txtxy job.415.log hello.py funcao_seno.png job.401.out a.out
script1.sh hpc_info.txt funcao_seno.png.png teste.slurm job.414.log python.py job.mpi
[mbanze@strge HPC_TRAINING]$ ./script2.sh
Escreva o seu nome
Ashley Martilio
Ola Ashley Martilio
[mbanze@strge HPC_TRAINING]$
```

Existem várias variáveis predefinidas contidas no Linux e RANDOM é uma delas. Agora vamos gerar um conjunto de 200 números aleatórios usando esta variável:

```
nano script3.sh
```



```
GNU nano 4.8                                script3.sh
#!/bin/bash

#Script used to generate a set of 200 random numbers

for numbers in {1..200}
do
echo $RANDOM
done

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line
```

Mesmo que a variável que eu uso para iterar de 1 a 200 seja chamada de números, essa variável nunca é usada no loop. Ele está lá apenas como um espaço reservado e o número que desejamos obter a cada iteração é um number RANDOM (número ALEATÓRIO).

Como este é um script recém-criado, é necessário modificá-lo antes de executá-lo:

```
chmod u+x script3.sh
```

```
./script3.sh
```

Você deve ter obtido monte de números aleatórios impressos na tela. Se quisermos verificar se de facto existem apenas 200 números aleatórios sendo impressos na tela, podemos fazer isso no terminal:

```
./script3.sh | wc -l
```

```
[mbanze@strge HPC_TRAINING]$ ./script3.sh | wc -l
200
[mbanze@strge HPC_TRAINING]$
[mbanze@strge HPC_TRAINING]$
```

Agora, em vez de obter os números aleatórios exibidos na tela, essas linhas são passadas para o wordcount e apenas o número total de linhas é impresso, verificando se temos 200 números aleatórios sendo gerados.

7. Exemplo de um job_script de Python:

[python_script](#)

8. Como usar o comando scp para transferir arquivos com segurança

Cópia segura (scp) é um utilitário de linha de comando que permite copiar arquivos e directórios com segurança entre dois locais.

Com scp, você pode copiar um arquivo ou directório:

- Do seu sistema local para um sistema remoto.
- De um sistema remoto para seu sistema local.
- Entre dois sistemas remotos do seu sistema local.

Ao transferir dados com scp, tanto os arquivos quanto a senha são criptografados para que qualquer pessoa que espione o tráfego não receba nada confidencial.

Neste tutorial, mostraremos como usar o comando scp através de exemplos práticos e explicações detalhadas das opções scp mais comuns.

8.1. Copiando arquivos e directórios entre dois sistemas com scp

Copie um arquivo local para um sistema remoto com o comando scp

Para copiar um arquivo de um sistema local para um remoto, execute o seguinte comando:

```
scp file.txt remote_username@siscad.morenet.ac.mz:/remote/directory
```

Onde file.txt é o nome do arquivo que queremos copiar, remote_username é o usuário no servidor remoto, siscad.morenet.ac.mz é o endereço IP (DNS) do servidor. O /remote/directory é o caminho para o directório para o qual você deseja copiar o arquivo. Se você não especificar um directório remoto, o arquivo será copiado para o directório inicial do usuário remoto.

Você será solicitado a inserir a senha do usuário e o processo de transferência será iniciado.

```

C:\Users\lenovo\Desktop>cd Shared_Folder

C:\Users\lenovo\Desktop\Shared_Folder>dir
Volume in drive C has no label.
Volume Serial Number is EC38-BCA1

Directory of C:\Users\lenovo\Desktop\Shared_Folder

07/20/2022  03:11 PM    <DIR>          .
07/20/2022  03:11 PM    <DIR>          ..
07/20/2022  02:52 PM             11,222 funcao.png.png
07/20/2022  03:10 PM             36,970 funcao_seno.png
07/20/2022  03:10 PM             36,970 funcao_seno.png.png
07/14/2022  06:32 PM    <DIR>          TCC
               3 File(s)             85,162 bytes
               3 Dir(s)  219,254,636,544 bytes free

C:\Users\lenovo\Desktop\Shared_Folder>scp funcao_seno.png.png mbanze@siscad.morenet.ac.mz:/home/mbanze/HPC_TRAINING
mbanze@siscad.morenet.ac.mz's password:
funcao_seno.png.png                                     100%  36KB 741.4KB/s   00:00

```

8.2. Para copiar um directório de um sistema local para remoto, use a opção -r:

```

C:\Users\lenovo\Desktop\Shared_Folder>scp -r \Users\lenovo\Desktop\Shared_Folder\TCC mbanze@siscad.morenet.ac.mz:/home/mba
nze/HPC_TRAINING
mbanze@siscad.morenet.ac.mz's password:
tcc_test-checkpoint.py                                     100% 6622   468.1KB/s   00:00
Untitled-checkpoint.ipynb                               100% 4723   444.5KB/s   00:00
Untitled1-checkpoint.ipynb                              100%  72    9.4KB/s     00:00
Untitled2-checkpoint.ipynb                              100%  72    8.6KB/s     00:00
Untitled3-checkpoint.ipynb                              100%  72    8.5KB/s     00:00
Untitled4-checkpoint.ipynb                              100%  72    7.6KB/s     00:00
2m_temp.png                                              100% 2396    87.2KB/s   00:00
grafico.py                                               100%  616    4.7KB/s     00:00
script_job.py                                            100%  860    7.9KB/s     00:00
tcc.py                                                   100% 3230    29.8KB/s   00:00
tcc_test.py                                              100% 7174   438.3KB/s   00:00
test.py                                                  100%  357    43.9KB/s   00:00
Untitled.ipynb                                           100% 4723   439.7KB/s   00:00
Untitled1.ipynb                                          100% 5624    50.5KB/s   00:00
Untitled2.ipynb                                          100%  589    68.6KB/s   00:00
Untitled3.ipynb                                          100% 110KB  690.0KB/s   00:00
Untitled4.ipynb                                          100%  96KB   4.8MB/s     00:00
uvief2021_world.nc                                     100% 1087MB  6.2MB/s     02:56

C:\Users\lenovo\Desktop\Shared_Folder>

```

8.3. Copiando um arquivo remoto para um sistema local usando o comando scp

```

C:\Users\lenovo\Desktop\Shared_Folder>scp mbanze@siscad.morenet.ac.mz:/home/mbanze/scratch/tcc.py .
mbanze@siscad.morenet.ac.mz's password:
tcc.py                                                  100% 3282   267.2KB/s   00:00

C:\Users\lenovo\Desktop\Shared_Folder>

```

ou

```
C:\Users>scp mbanze@siscad.morenet.ac.mz:/home/mbanze/HPC_TRAINING C:\Users\lenovo\Desktop\Shared_Folder\funcao_seno.png
mbanze@siscad.morenet.ac.mz's password:
scp: /home/mbanze/HPC_TRAINING: not a regular file

C:\Users>scp mbanze@siscad.morenet.ac.mz:/home/mbanze/HPC_TRAINING/funcao_seno.png C:\Users\lenovo\Desktop\Shared_Folder
mbanze@siscad.morenet.ac.mz's password:
funcao_seno.png          100%  11KB  95.4KB/s   00:00

C:\Users>
```

8.4. Copiando um arquivo entre dois sistemas remotos usando o comando scp

Ao usar o scp você não precisa fazer login em um dos servidores para transferir arquivos de uma máquina remota para outra.

O comando a seguir copiará o arquivo /files/file.txt do host remoto host1.com para o directório /files no host remoto host2.com.

```
scp user1@host1.com:/files/file.txt user2@host2.com:/files
```

Você será solicitado a inserir as senhas para ambas as contas remotas. Os dados serão transferidos directamente de um host remoto para outro.