

# Preparação do Ambiente

---

Advanced Institute for Artificial Intelligence – AI2

<https://advancedinstitute.ai>



# Sistemas Unix

---

- ❑ <https://datasciencepractice.study/>
- ❑ <https://www.datascienceatthecommandline.com/>
- ❑ <https://swcarpentry.github.io/shell-novice>
- ❑ [http://people.duke.edu/~ccc14/duke-hts-2018/cliburn/The\\_Unix\\_Shell\\_02\\_03\\_Working\\_with\\_Text.html](http://people.duke.edu/~ccc14/duke-hts-2018/cliburn/The_Unix_Shell_02_03_Working_with_Text.html)
- ❑ <https://www.lpi.org/>

# Agenda

1. Introdução/Motivação
2. Interação com o SO - Comandos Básicos
3. Gerenciando Processos
4. Manipulação de Arquivos

## Motivação

Como cientista de dados, **é quase impossível evitar os sistemas Unix**. Unix vs Windows não é uma questão de preferência como R vs Python; se você não conhece o Unix, **é provável que tenha dificuldades no local de trabalho**. Unix é o **sistema operacional da ciência de dados**, então você precisa saber como usá-lo.

## O que é Unix?

“

*Os sistemas Unix são caracterizados por um **design modular** que às vezes é chamado de “filosofia Unix”. Este conceito implica que o sistema operacional fornece um conjunto de **ferramentas simples**, cada uma desempenhando uma **função limitada e bem definida**, com um sistema de arquivos unificado (o sistema de arquivos Unix) como principal meio de comunicação, e script shell e linguagem de comando (o Unix shell) **para combinar as ferramentas** para realizar **fluxos de trabalho complexos**.*

Wikipedia

## Cenários de Uso

Você pode encontrar sistemas Unix como um cientista de dados:

- ☐ ao trabalhar na linha de comando usando sistemas operacionais macOS ou Linux,
- ☐ ao trabalhar com servidores remotos no data center do seu empregador,
- ☐ ao trabalhar com servidores remotos fornecidos pela AWS, Azure, GCP, etc,
- ☐ ao trabalhar com servidores Jupyter ou RStudio compartilhados (eles são hospedados em servidores Unix)

## Cenários de Uso

Alguns exemplos de uso do Unix para ciência de dados incluem:

- ☐ usando `ssh` para se conectar a um servidor de análise remoto
- ☐ usando `scp` para mover arquivos entre servidores
- ☐ usando um gerenciador de pacotes (por exemplo, `brew`, `apt`, `yum`) para instalar ferramentas de linha de comando
- ☐ usando ferramentas de linha de comando para interagir com serviços de terceiros
- ☐ usando `wget` para baixar arquivos da internet
- ☐ usando `top` para monitorar a utilização de recursos do sistema
- ☐ usando `docker` para gerenciar e executar contêineres

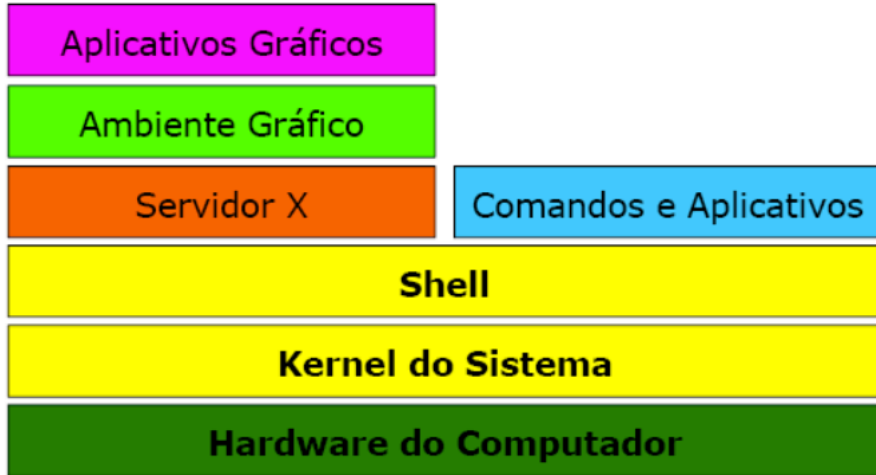


O Sistema Operacional (SO) é o programa que controla o computador, servindo de Interface entre o usuário e a máquina. O Sistema Operacional faz isso através de dois componentes: O Kernel e o Shell

- ❑ Kernel é o nome dado ao núcleo do Sistema Operacional. É o módulo deste programa que se comunica com o hardware do computador
- ❑ Shell é a “fachada” do Sistema Operacional. Essa é a parte do programa que se comunica com o usuário, recebendo seus comandos e repassando-os ao Kernel

## Distribuição Linux

- É o nome dado ao conjunto de programas formado pelo Kernel Linux e por mais alguns softwares distintos (como Shells, aplicativos, jogos, utilitários, etc.)
- Várias empresas (ou pessoas) podem agrupar os programas que acham interessantes e criar suas próprias distros
- O Que Há Numa Distribuição?
  - Kernel, shell e ambiente gráfico
    - KDE, Gnome, ABlackBox, WindowMaker, Fluxbox



- Protege usuários de lidar com as *entranhas* do SO;
- Quando as pessoas dizem "shell" hoje em dia, quase sempre se referem ao **B**ourne **A**gain **S**hell (**bash**)
- Exemplo de um REPL - *Read-Evaluate-Print Loop*.
  1. o shell lê seu comando;
  2. o shell avalia seu comando e calcula a saída;
  3. o shell imprime a saída para o terminal;
  4. o shell imprime um novo prompt, pronto para seu próximo comando;

## Aplicações para o Shell

Trata-se de um programa de usuário, que oferece uma interface personalizável para o sistema e ainda:

- ☐ Interface de texto e interativa (intérprete de comandos);
- ☐ Ambiente de execução de outros programas;
- ☐ Linguagem de programação onde, programas completos podem ser escritos;
- ☐ São personalizáveis, podendo-se ajustar ao ambiente operacional;

## Exemplos de Shells

Trata-se de um programa de usuário, que oferece uma interface personalizável para o sistema e ainda:

- ☐ sh;
- ☐ bash;
- ☐ ksh;
- ☐ zsh;
- ☐ csh;
- ☐ tcsh;

## Exemplos de Shells

Trata-se de um programa de usuário, que oferece uma interface personalizável para o sistema e ainda:

- ☐ sh;
- ☐ bash;
- ☐ ksh;
- ☐ zsh;
- ☐ csh;
- ☐ tcsh;

Podemos instalar essas versões:

- ☐ sudo apt-get install **nome-do-shell**

## Aplicações para o Shell

É possível verificar quem está executando e qual shell esta sendo executado, no caso do Ubuntu será o `/bin/bash` :

- ☐ `echo $SHELL`
- ☐ `whoami`



## Variáveis de Ambiente

São variáveis utilizadas por programas e também por outros shells, sendo distribuídas em dois tipos de variáveis, Globais e Locais:

### □ Globais:

1. **PATH:** Lista os diretórios dos programas executáveis;
2. **USERNAME:** Nome do usuário logado;
3. **TERM:** Tipo do terminal em uso;
4. **HOME:** Diretório home do usuário atual.

## Variáveis de Ambiente

As variáveis locais, ou também variáveis do Shell, são específicas para cada shell aberto, possuindo também seus valores particulares:

□ Locais:

1. **SECONDS**: Número de segundos que o Shell está aberto;
2. **SHELL**: Apresenta informação sobre o shell em uso;

Podemos também criar variáveis locais para o shell em uso: **teste=valor1** Para apagar (desalocar da memória essa variável, utilizamos: ) **unset teste**

Comandos de sessão:

- ☐ Login: iniciar sessão: Interface Gráfica ou Terminal
- ☐ Logoff : sair da sessão
  - ☐ `exit`
- ☐ Reboot : reinicia o sistema
  - ☐ `reboot`
- ☐ `poweroff` ou `shutdown -h now` desliga o sistema

Atalhos úteis em uma sessão:

- ☐ TAB auto completa comandos
- ☐ History mostra a lista de comandos executados
- ☐ ! id-histórico executa o comando do histórico com o id id-histórico
- ☐ Man mostra ajuda de comandos

## Estrutura de arquivos

- ❑ Referência a um arquivo/diretório é feita através de um caminho, que é formado da seguinte forma:
- ❑ [diretório raiz] [diretório 1] [diretório 2] ... [diretório n] [arquivo]
- ❑ Cada um desses itens são separados por uma / (barra) no Linux
- ❑ o diretório raiz chama-se / (barra)
  - Exemplo de caminho: `/home/rmcobe/doc/1`
- ❑ exemplo de caminho: `/home/rmcobe/doc/1`

## Comandos Básicos - Manipulação do Sistema de Arquivos

- ☐ **cd**: Comando para acessar um diretório
- ☐ **pwd**: retorna o caminho do diretório desde a raiz /
- ☐ **cp**: copia um arquivo ou diretório de uma caminho para outro
- ☐ **find**: procurar por arquivos ou diretórios no sistema de arquivos
- ☐ **mkdir**: cria um novo diretório
- ☐ **mv**: move um arquivo ou diretório de um caminho para outro
- ☐ **ls**: lista um diretório

## Comandos Básicos - Manipulação do Sistema de Arquivos

- ☐ **rm**: remove um diretório, desde que esteja vazio
- ☐ **touch**: cria um arquivo vazio
- ☐ **du**: retorna o espaço utilizado por um diretório ou arquivo
- ☐ **df**: retorna as partições presentes no sistema
- ☐ **tree**: retorna a árvore do sistema
- ☐ **chmod**: muda a permissão de arquivos e diretórios

## Alguns diretórios possuem notações especiais ou "atalho"

- `~`: ao referir-se ao diretório `~` (til), o sistema entende como o diretório pessoal do usuário, ou seja, `/home/[usuário]`, onde `[usuário]` é o nome de login do usuário atual.
- `-`: o kernel Linux armazena um histórico dos diretórios que acessamos. O `-` (hífen) refere-se ao último diretório acessado
- `.`: o símbolo `.` (ponto) refere-se ao diretório atual, ou seja, aquele em que estamos trabalhando
- `..`: o `..` (ponto ponto) refere-se ao diretório acima do qual estamos.



No Linux existem três tipos de permissão para definir os acessos a arquivos e diretórios:

- ☐ **r** : Permissão de leitura para um arquivo; e permitir listar o conteúdo de um diretório através do comando `ls` diretório
- ☐ **w** : Permissão de gravação e exclusão para um arquivo/diretório.
- ☐ **x** : Permissão para executar um arquivo, se for um arquivo binário ou um script; e se for um diretório permitir acesso a ele através do comando `cd` diretório
- ☐ Tais permissões são concedidas aos seguintes papéis de usuário
  - **Dono** : O proprietário do arquivo ou criador do arquivo
  - **Grupo**: Usuários que fazem parte do grupo do proprietário
  - **Outros**: Não são os proprietários e nem fazem parte do grupo

# Manipulando Arquivos



Quando o comando contém a letra 'd' na primeira coluna, trata-se de um diretório!!!

**Exemplo:** drwxr-xr-x 28 clayton clayton

# Permissões para usuários no Arquivo

Um breve exemplo dos valores de permissões:

```
1 drwxr--r-- 1 clayton      clayton      8704 set 16 20:58 |
2 | | | | | |
3 | | | | | \-----> Todos os usuários podem ler o interior do diretório, mas não
4 | | | | |      podem usar o comando 'cd' para entrar nele, pois não existe
5 | | | | |      o FLAG 'x' para a quarta coluna!
6 | | | | \-----> Usuarios do grupo 'root' podem ler o interior do diretório,
7 | | | |      mas também não podem usar 'cd' para entrar no diretório!
8 | | | \-----> O usuário 'root' pode usar 'cd' para entrar no diretório!
9 | | \-----> O usuário 'root' pode gravar arquivos nesse diretório!
10 | \-----> O usuário 'root' pode ler o interior desse diretório!
11 \-----> Indica que o nome listado é um diretório!
```

O comando `chmod` também pode ser usado com números, em vez das *flags*:

**`chmod 664 aula01.sh`**

Mas o que quer dizer cada um desses números? Vejamos:

- ☐ 0 : nenhuma permissão;
- ☐ 1 : permissão para executar;
- ☐ 2 : permissão para gravar;
- ☐ 3 : permissão para gravar e executar;
- ☐ 4 : permissão para ler;
- ☐ 5 : permissão para ler e gravar;
- ☐ 6 : permissão para ler, gravar e executar.

## Filtros

- ❑ **grep** Examina cada linha de dados que recebe da entrada padrão e produz cada linha que contém um padrão especificado de caracteres.
- ❑ **sort** Classifica a entrada padrão e produz o resultado classificado na saída padrão
- ❑ **uniq** Dado um fluxo de dados classificado da entrada padrão, ele remove linhas de dados duplicadas (ou seja, garante que cada linha seja única).
- ❑ **head** Exibe as primeiras linhas de sua entrada. Útil para obter o cabeçalho de um arquivo.
- ❑ **tail** Exibe as últimas linhas de sua entrada. Útil para coisas como obter as entradas mais recentes de um arquivo de log.

## Variáveis de Ambiente

- ☐ **export** : cria uma variável
- ☐ **env** : mostra as variáveis criadas
- ☐ **unset** : apaga uma variável
- ☐ **echo** : mostra o valor atribuído a uma variável

Pesquisando processos em execução comando ps

- ☐ Exibe informações sobre os processos ativos
- ☐ `ps [opções]`
- ☐ `a` exibe também informações de outros usuários
- ☐ `u` exibe o nome do usuário e a hora de início do processo
- ☐ `x` exibe também os processos não associados a um terminal de controle
- ☐ `-p pid` exibe o processo cujo número é pid

kill: finaliza um processo por meio do pid;

- ❑ `kill [opções] [sinal] pid`
- ❑ `-n` sinal aplicado ao processo
- ❑ `-l` lista todos os nomes e números de sinais
  - `kill -9 1029`



Comandos para exibir informações sobre o computador de forma geral

- ❑ **free**: exibe a quantidade de memória livre;
- ❑ **lscpu**: exibe informações sobre o CPU

## Redirecionamento

- Redirecionamento de saída:
  - `>` e `>>`
- Redirecionamento de entrada:
  - `<` e `<<`
  - `sort < arquivo`
- Pipe (`|`) - redireciona a saída de um comando para a entrada do próximo:
  - `cat arquivo | uniq`

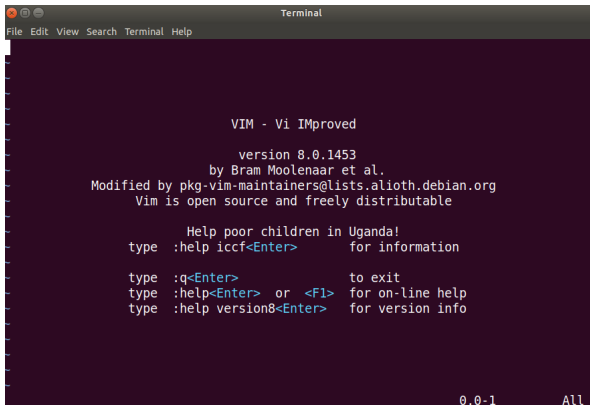
**cat** : concatena arquivos e lista na saída padrão

- ❑ Sintaxe: **cat** [opções] parametros
- ❑ Parâmetros pode ser uma lista contendo arquivos
- ❑ Exemplo de uso: **cat** arq1.txt arq2.txt
  - Concatena arq1.txt e arq2.txt e exibe na saída padrão
- ❑ Outro Exemplo: **cat** > arq3.txt

# Arquivos de Script

Para criar um arquivo de scripts, podemos usar qualquer editor como **vi**, **nano** e **gedit**.

A combinação **#!** é o que indica ao shell que o arquivo se trata de um script.



```
Terminal
File Edit View Search Terminal Help

VIM - Vi IMproved

version 8.0.1453
by Bram Moolenaar et al.
Modified by pkg-vim-maintainers@lists.alioth.debian.org
Vim is open source and freely distributable

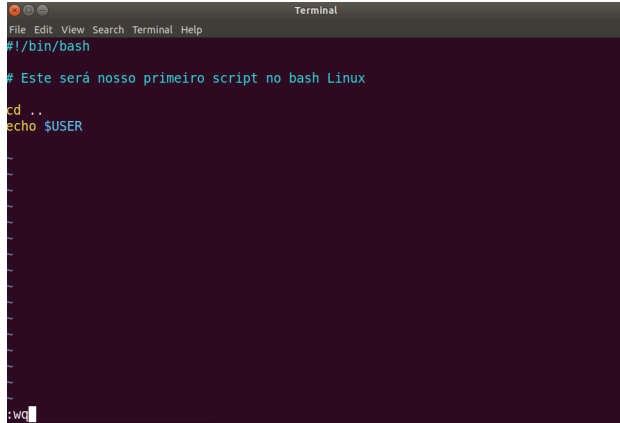
Help poor children in Uganda!
type :help iccf<Enter>      for information

type :q<Enter>              to exit
type :help<Enter> or <F1>   for on-line help
type :help version8<Enter> for version info

0,0-1 All
```

# Arquivos de Script

No **vi**, com o comando "i" de **insert**, podemos iniciar o processo de edição e criação do nosso script.

A screenshot of a macOS Terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal content shows a shell prompt "#!/bin/bash" followed by a comment "# Este será nosso primeiro script no bash Linux". Below the comment, the user has entered "cd .." and "echo \$USER". The terminal is in vi editor mode, with the cursor at the end of the "echo \$USER" line. The bottom of the terminal shows the path ":wq" and a cursor, indicating the user is in command mode.

```
#!/bin/bash

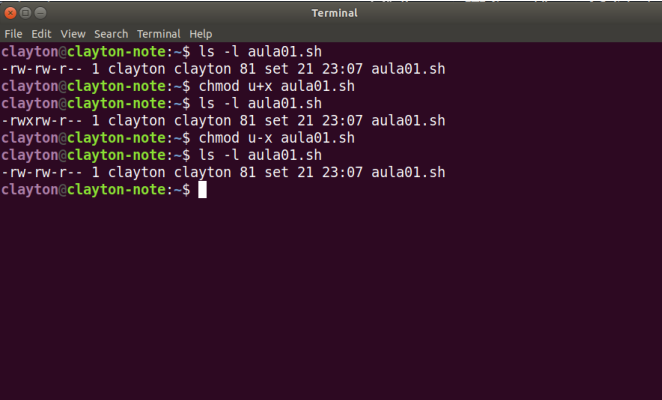
# Este será nosso primeiro script no bash Linux

cd ..
echo $USER

:wq
```

# Arquivos de Script

Para alterar as permissões do arquivo, utilizamos o comando **chmod** e os valores desejados:

A screenshot of a macOS Terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal shows a series of commands and their outputs. The user is at the prompt "clayton@clayton-note:~". The commands and outputs are: 1. "ls -l aula01.sh" outputs "-rw-rw-r-- 1 clayton clayton 81 set 21 23:07 aula01.sh". 2. "chmod u+x aula01.sh" outputs nothing. 3. "ls -l aula01.sh" outputs "-rwxrw-r-- 1 clayton clayton 81 set 21 23:07 aula01.sh". 4. "chmod u-x aula01.sh" outputs nothing. 5. "ls -l aula01.sh" outputs "-rw-rw-r-- 1 clayton clayton 81 set 21 23:07 aula01.sh". The prompt "clayton@clayton-note:~\$" is shown at the end of the last output line.

```
clayton@clayton-note:~$ ls -l aula01.sh
-rw-rw-r-- 1 clayton clayton 81 set 21 23:07 aula01.sh
clayton@clayton-note:~$ chmod u+x aula01.sh
clayton@clayton-note:~$ ls -l aula01.sh
-rwxrw-r-- 1 clayton clayton 81 set 21 23:07 aula01.sh
clayton@clayton-note:~$ chmod u-x aula01.sh
clayton@clayton-note:~$ ls -l aula01.sh
-rw-rw-r-- 1 clayton clayton 81 set 21 23:07 aula01.sh
clayton@clayton-note:~$
```

Como podemos ver, o `+` ou `-` definem se os *FLAGS* serão ativados ou desativados!

Temos ainda outros exemplos:

- ❑ `chmod a+r aula01.sh`: Todos usuários (`a=all`) podem ler o arquivo;
- ❑ `chmod o+w aula01.sh`: Outros usuários (`o=others`) sem ser o dono e o grupo dono do arquivo, podem gravar o arquivo;
- ❑ `chmod g+x aula01.sh`: O grupo-dono do arquivo (`g=group`) pode executar o arquivo.

# Vamos Exercitar???

**Um arquivo chamado Aula01\_manipulação.txt contém alguns exercícios baseados no conteúdo dessa nossa aula.**

**Bom trabalho!!!**