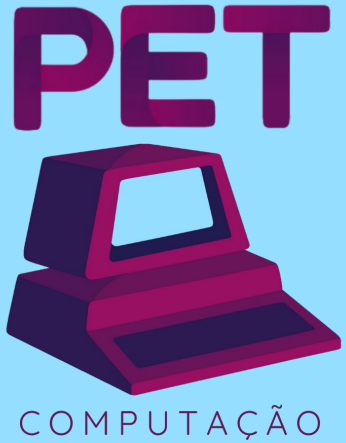
Linux

Aula III



Login: clinux2024X
Senha: Clinux2024#X

2024

Relembrando...

Comandos

- `cd` - altera o diretório atual (*working directory*)
- `ls` - lista conteúdo do diretório
- `touch` - atualiza data de acesso/modificação ou cria novo arquivo
- `mkdir` - cria diretório
- `rm & rmdir` - remove arquivo/diretório
- `cp` - copia arquivo/diretório
- `mv [arquivo] [destino]` - move arquivo/diretório
- `mv [arquivo] [novo nome]` - renomeia arquivo/diretório
- `file` - especifica tipo do arquivo
- `du` - mostra o tamanho de arquivo/diretório
- `quota` - mostra o consumo de sua quota na macalan

Texto para treino

wget https://www.inf.ufpr.br/dlpg21/linux/memorias.txt

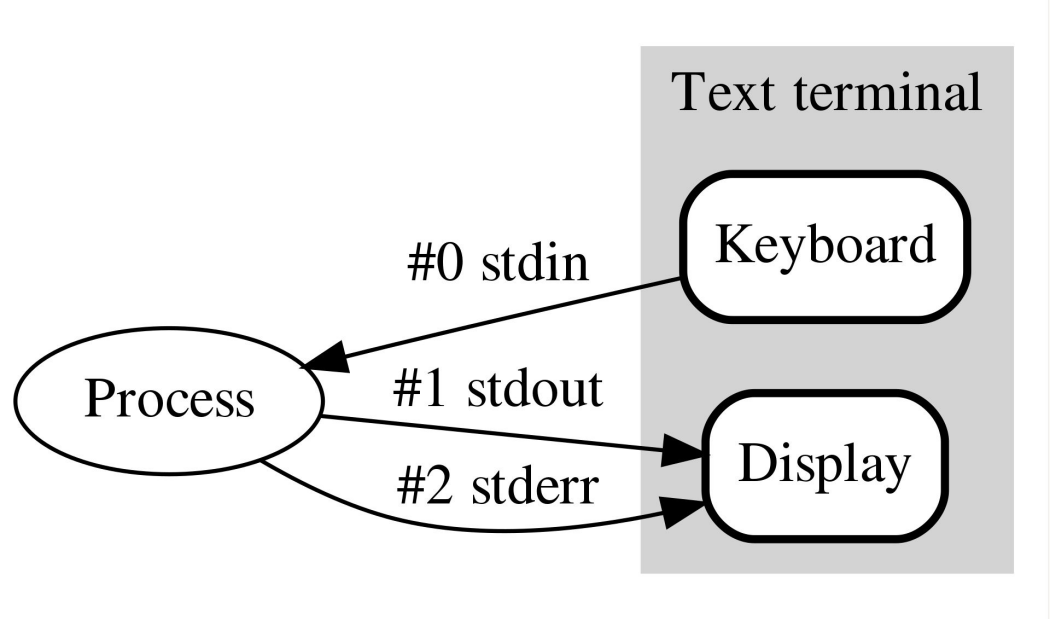
1

Entrada e saída de dados

Streams

- Nos sistemas Unix, há a **convenção** de utilização de entradas e saídas padrão para a entrada e saída de dados
- São definidas **três conexões**/*streams*/fluxos padrão:
  - *stdin* → **st**an**d**ard **in**put (entrada padrão)
  - *stdout* → **st**an**d**ard **out**put (saída padrão)
  - *stderr* → **st**an**d**ard **err**or (erro padrão)
- Por estas streams ocorre o fluxo dados entre o teclado, processos e a tela, mediados pelo shell
- Por padrão, o *stdin* recebe dados do teclado, e *stdout* e *stderr* imprime mensagens na tela
- Contudo, é possível utilizar conteúdo de arquivos no *stdin*, e redirecionar a saída de *stdout* e *stderr* para arquivos, em vez da tela
- Então, a partir daqui, em vez de usarmos a expressão “imprimir na tela”, falaremos “escrever na saída/erro padrão”
- Veremos como redirecionar as *streams* adiante

Streams


Text terminal
Keyboard
#0 stdin
Process
#1 stdout
Display
#2 stderr

echo

- Escreve o texto (argumento) na saída padrão
- Por padrão, escreve o caractere de nova linha `\n` no fim do texto
- Para evitar a escrita do *new line*, utilize a opção **-n**

```
[pet@arch ~]$ echo 'Hello, world!'
Hello, world!
[pet@arch ~]$ echo -n 'Hello, world!'
Hello, world![pet@arch ~]$
```

cat & **tac** con**cat**enate

- Lê arquivo(s) passado(s) como argumento(s), o(s) concatena e escreve na saída padrão
- Caso não seja passado nenhum argumento, lê os dados da entrada padrão (*stdin*)
- O comando **tac** performa a mesma tarefa do **cat**, porém, escreve as linhas na ordem reversa

```
[pet@arch ~]$ cat hello.txt
Hello, world!
[pet@arch ~]$ cat test.c
#include <stdio.h>

int main() {
    return 0;
}
[pet@arch ~]$ cat hello.txt test.c
Hello, world!
#include <stdio.h>

int main() {
    return 0;
}
[pet@arch ~]$
```

```
[pet@arch ~]$ tac hello.txt
Hello, world!
[pet@arch ~]$ tac test.c
}
    return 0;
int main() {

#include <stdio.h>
[pet@arch ~]$ tac hello.txt test.c
Hello, world!
}
    return 0;
int main() {

#include <stdio.h>
[pet@arch ~]$
```

2.

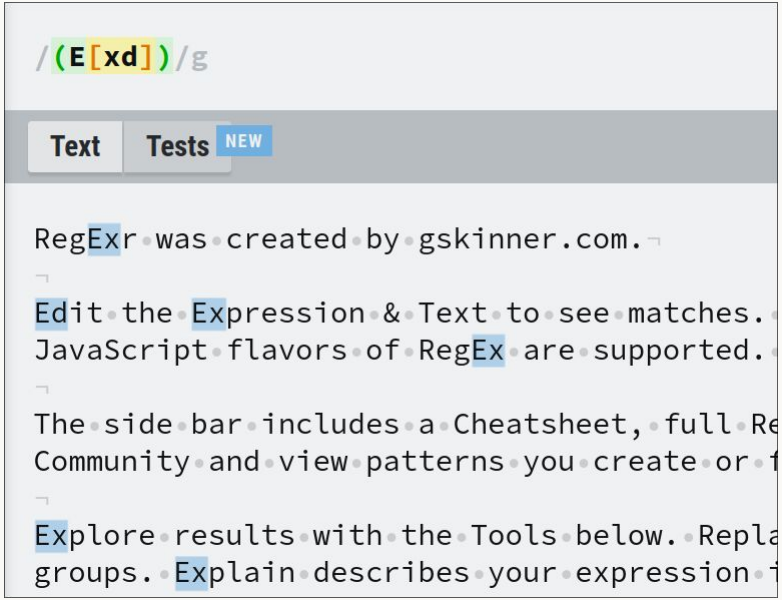
Caracteres coringas

Redirecionamento das *streams*

- É possível alterar a fonte da *stdin* e as saídas de *stdout* e *stderr* a partir de caracteres coringas
- Para redirecionar a saída padrão para um arquivo, utiliza-se **>** e **>>**, destrutivo e não-destrutivo respectivamente
  - `echo ‘Hello, world!’ > hello.txt`
  - `echo ‘abc’ >> hello.txt`
  - `echo ‘destruindo’ > hello.txt`
  - `echo ‘echo abc’ > echo.txt`
- Para enviar dados de um arquivo para *stdin*, utiliza-se **<**
  - `cat < hello.txt`
  - `bash < echo.txt`

Regex regular expressions

- As expressões regulares são amplamente utilizadas em diversas áreas da computação
- Com elas, é possível representar **padrões** de texto a partir de sequência de caracteres
- **Alguns** de seus caracteres são possíveis de serem usados no shell como caracteres coringas
- 💡 Pesquise sobre **regex**



regexr.com

Caracteres coringas

- **`*`** → representa zero ou mais ocorrências de quaisquer caracteres
- **`?`** → representa uma ocorrência de um caractere qualquer, pode ser combinado (ex: **`?????.txt`**)
- **`[]`** → representa uma ocorrência de algum caractere presente entre os colchetes, pode ser combinado (ex: **`[2468][13579].txt`**)
- **`{}`** → com chaves é possível representar sequências de caracteres
  - `echo {1..100}`
  - `echo {a..z}`
- & → colocado no fim de um comando, coloca o processo em *background* e libera o prompt
  - gedit teste.txt &

Caracteres coringas

- ; → com este caracter é possível separar comandos em apenas uma linha
  - `echo a; echo b`
- **$** → usada para representar variáveis de ambiente do shell
  - `echo $PATH`
- “ → usada para representar texto permitindo caracteres especiais
  - `echo “Data: $(date)”`
- ‘ → usada para representar texto anulando efeito dos caracteres especiais
  - `echo ‘Data: $(date)’`

Caracteres coringas

```
[pet@arch test]$ touch ts{1..5}.txt
[pet@arch test]$ ls
ts1.txt  ts2.txt  ts3.txt  ts4.txt  ts5.txt
[pet@arch test]$ touch tr{1..5}.txt
[pet@arch test]$ ls
tr1.txt  tr3.txt  tr5.txt  ts2.txt  ts4.txt
tr2.txt  tr4.txt  ts1.txt  ts3.txt  ts5.txt
[pet@arch test]$ ls tr*.txt
tr1.txt  tr2.txt  tr3.txt  tr4.txt  tr5.txt
[pet@arch test]$ ls t?[1-3].txt
tr1.txt  tr2.txt  tr3.txt  ts1.txt  ts2.txt  ts3.txt
```

3.

Variáveis de ambiente

Variáveis de ambiente

- Variáveis são objetos (localizadas na memória) que armazenam valores para serem usados em um programa
- A shell também possui variáveis, as quais podem ser alteradas pelo usuário, são chamadas de **variáveis de ambiente** (*environment variables*) ou **variáveis shell**
- Os nomes destas variáveis iniciam com **$**, e são escritas em letras maiúsculas
- Exemplos:
  - $HOME → armazena o diretório pessoal do usuário (ex: “/home/pet”)
  - $USER → armazena o usuário logado que abriu a shell (ex: “pet”)
  - $LANG → armazena a linguagem do sistema operacional (ex: “pt_BR.UTF-8”)
  - $SHELL → armazena a localização do executável da shell atual (ex: “/bin/bash”)

Variáveis de ambiente

- Para ler uma variável de ambiente específica, utilize o comando **echo**

```
dlpg21@macalan:~$ echo $HOME
/home/bcc/dlpg21
dlpg21@macalan:~$ echo $USER
dlpg21
dlpg21@macalan:~$ echo $LANG
pt_BR.UTF-8
dlpg21@macalan:~$ echo $SHELL
/bin/bash
```

Variáveis de ambiente

- Para listar todas as variáveis de ambiente, utilize o comando **printenv**

```
dlpg21@macalan:~$ printenv
SHELL=/bin/bash
LMOD_arch=x86_64
NOBACKUP=/nobackup/bcc/dlpg21
XDG_DATA_HOME=/nobackup/bcc/dlpg21/.local/share
LMOD_DIR=/usr/share/lmod/lmod/libexec
PWD=/home/bcc/dlpg21
LOGNAME=dlpg21
XDG_SESSION_TYPE=tty
MODULESHOME=/usr/share/lmod/lmod
MANPATH=/usr/share/lmod/lmod/share/man:::/home/soft/likwid/man
LMOD_PREPEND_BLOCK=normal
MOTD_SHOWN=pam
HOME=/home/bcc/dlpg21
LANG=pt_BR.UTF-8
```

Variável $PATH

- A variável $PATH armazena os diretórios, separados pelo caractere “:”, nos quais a shell procurará os executáveis dos comandos digitados pelo usuário
- Por exemplo, ao utilizar o comando **ls**, a shell precisará encontrar seu arquivo executável, assim, procurará em todos os diretórios de $PATH até encontrá-lo, que então será executado
- Logo, caso você precise utilizar um programa que não esteja em um diretório da $PATH, precisará utilizar o início “./”, seguido do caminho do arquivo

Variável $PATH

- Exemplo de conteúdo da $PATH

```
dlpg21@macalan:~$ echo $PATH | tr ":" "\n"
/home/bcc/dlpg21/.local/bin
/home/bcc/dlpg21/bin
/home/bcc/dlpg21/.cargo/bin
/usr/local/bin
/usr/bin
/bin
/usr/local/games
/usr/games
/usr/local/sbin
/usr/sbin
/sbin
/home/soft/likwid/bin
/home/soft/likwid/sbin
```

* O comando tr substitui todas as ocorrências de um caractere por outro

Adicionar diretório ao $PATH

- É possível adicionar diretórios à variável $PATH, para isso utilize o comando **export**
- `export PATH="<diretório>:$PATH"`

```
dlpg21@macalan:~$ ls abc
helloworld  helloworld.c
dlpg21@macalan:~$ helloworld
helloworld: command not found
dlpg21@macalan:~$ export PATH="/home/bcc/dlpg21/abc:$PATH"
dlpg21@macalan:~$ helloworld
Hello, world!
```

Adicionar diretório ao $PATH

- Contudo, esta mudança é temporária, e não funcionará em outras sessões shell
- Para tornar a mudança permanente, é preciso adicionar o comando ao arquivo de configuração da shell, o “~/.bashrc” ou “~/.profile” para o bash
- Toda vez que uma shell bash é iniciada, estes arquivos são executados
- Logo, ao iniciar a shell, o comando export será executado, atualizando o $PATH

4.

Usuários e grupos

Usuários e grupos

- Usuários e grupos são usados para controle de acesso a arquivos e diretórios
- Usuários podem ser adicionados e removidos, com eles é possível logar no sistema
- Grupos são conjuntos de usuários, que podem receber permissões específicas que se aplicam a todos os usuários presentes nele
- Ao instalar o Linux, um usuário é criado por padrão, o usuário **root**

Superusuário (sudo)

- O root é um **superusuário**, ou seja, é um usuário administrador que **pode controlar qualquer aspecto do sistema operacional**, desde acessar qualquer arquivo/diretório a rodar qualquer comando
- Como usuário comum, é possível rodar um comando como root usando o comando **sudo (Super User DO!)** como prefixo
  - `$ sudo apt install nvim`
- Contudo, para poder utilizar o comando sudo, o usuário deve estar no grupo sudoers, que apenas o **root** e outros usuários sudoers conseguem gerenciar
  - `$ cat /etc/sudoers` (só conseguirá rodar como superusuário, rs)

5.

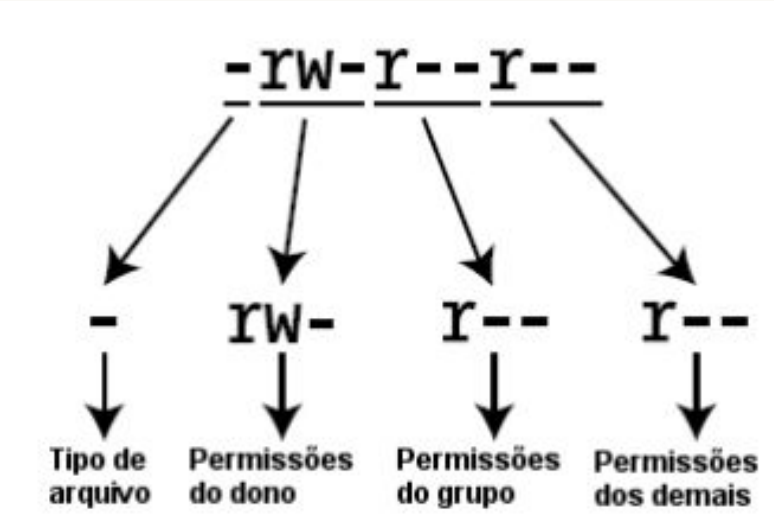
Permissões

Permissões

- Há três tipos de permissões aplicadas a arquivos e diretórios:
  - **R**ead: leitura
  - **W**rite: escrita, alteração e deleção
  - E**x**ecute: execução
- Estas permissões podem ser definidas para três entidades:
  - **U**ser: seu usuário (dono do arquivo/diretório)
  - **G**roup: usuários presentes nos seus grupos
  - **O**thers: o resto
- Cada arquivo e diretório possuem um **dono**, que é o criador do item
- O dono pode gerenciar seus arquivos e diretórios para atribuir permissões de leitura, escrita e/ou execução a outros usuários
- O superusuário tem permissão para leitura, escrita e execução em **qualquer arquivo/diretório!**

Permissões

- Rode: `$ ls -l`




```
$ ls -l
drwxr-xr-x 2 chapolin pet-user 4096 2011-02-25 16:22 Comandos
-rw-r--r-- 1 chapolin pet-user  392 2011-02-25 14:37 Makefile
```

chmod change mode

- Para alterar permissões, utiliza-se o comando chmod
- O primeiro argumento é as entidades em que as permissões serão mudadas
- O segundo argumento é o tipo de alteração
- O terceiro argumento é o tipo de permissão
- A notação textual segue:
  - `$ chmod [ugoa] [+-=] [rwx] [arquivo/diretório]`
- As possibilidades do terceiro argumento podem ser combinadas
  - **Ex:** `$ chmod ug=rw, o-rw hello.txt`
- Ao ser utilizado em diretórios, é possível utilizar a opção -R (**R**ecursivo) para aplicar as mudanças em todos os arquivos/diretórios filhos
- 💡 Pesquise sobre a notação octal do **chmod**

public_html

- Ao adicionar um arquivo no seu diretório public_html do DInf, não se esqueça de alterar as permissões para permitir a leitura e execução do arquivo pelo servidor nginx
- `$ chmod o+rx index.html`

6.

Histórico

history

- A shell, por padrão, escreve os comandos executados num arquivo de histórico
- Bash: ~/.bash_history
- zsh: ~/.zsh_history
- Assim, é possível utilizar o comando **history** que exibe o histórico dos comandos
- Para limpar o histórico, utilize:
  - `history -c`

7.

Redirecionamento de streams entre comandos

Pipeline

- Você pode redirecionar a saída de um comando (*stdout*) para ser usada como entrada (*stdin*) de outro usando **pipe** (caractere '|')
- `comando1 | comando2 | comando3` - comando1 gera entrada pro comando2 e o comando2 pro comando3.
- Exemplos:
  - `ls | tac` - imprime o **ls** da última linha para a primeira
  - `ls -t | tac` - imprime o **ls** em ordem crescente de data

8.

Mais comandos de leitura

head

- Por padrão, lê as **primeiras** 10 linhas de um arquivo
- É possível especificar quantas linhas lidas com a opção `-n [número]`
- Ex:
  - `$ head arquivo.txt`
  - `$ head -n 20 arquivo.txt`

tail

- Por padrão, lê as **últimas** 10 linhas de um arquivo
- É possível especificar quantas linhas lidas com a opção `-n [número]`
- Ex:
  - `$ tail arquivo.txt`
  - `$ tail -n 20 arquivo.txt`

less

- Lê um arquivo de **forma interativa**
- Isto é, não imprime todo o arquivo na tela, em vez disso, é possível navegar pelo arquivo utilizando setas, mouse etc.
- Ideal para arquivos grandes
- “q” para sair, “h” para mostrar a tela de ajuda
- “/padrao” **grifa as ocorrências** de “padrao” no documento
- Versão melhorada do comando “more”

9.

Buscando arquivos

find

- Formato: **find** `[ponto-inicial] [opções] [argumento]`
- Ponto inicial: por onde começar a procura (padrão: diretório atual)
- Opções: no geral, especifica que tipo de argumento o comando irá procurar, como nome, data de criação, permissão. Exemplo de opção: -iname.
- Argumento: o que o comando irá procurar levando em consideração o tipo dado nas opções.

10.

Filtrando texto

grep

- Procura por uma palavra/frase/expressão dentro de um arquivo.
  - **G**lobal **R**egular **E**xpression **P**rint
  - **G**lobally looks for a **R**egular **E**xpression and **P**rint
  - **G**lobal **R**egular **E**xpression **P**arser
- Formato: **grep** `[opções][expressão][arquivo]`
- `cat doc.txt | grep legal -` Pega as linhas de **doc.txt** com a palavra “legal”
- Você pode sempre usar o man para ver as opções:
  - `man grep`

cut

- Usado para “cortar” o texto por colunas
- Opções
  - `-d'c'`: usa ‘**c**’ como caractere delimitador
  - `-fn` : retorna apenas o campo (*field*) número **n**
  - `-fn,m` : retorna apenas os campos (*field*) número **n e m**
- O caractere TAB é o padrão de delimitador
- Parecido com outro comando: **awk**
- `cat documento.csv | cut -d',' -f1 -` Colunas separadas por vírgula, coluna 1

Exercício

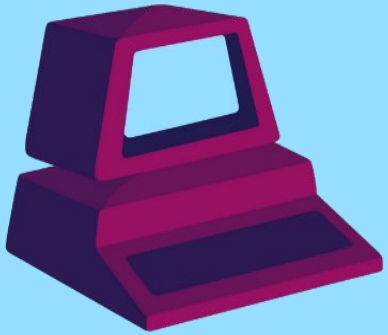
```
$ wget https://www.inf.ufpr.br/dlpg21/linux/aula2.tar.gz
$ tar -xvf aula2.tar.gz

$ wget https://www.inf.ufpr.br/dlpg21/linux/aula3.tar.gz
$ tar -xvf aula3.tar.gz
```

Avalie a aula

forms.gle/T9SXjgBs2pyt9DJg6

Conta como presença!

Obrigado!



PET
COMPUTAÇÃO

pet.inf.ufpr.br

pet@inf.ufpr.br

@petcompufpr