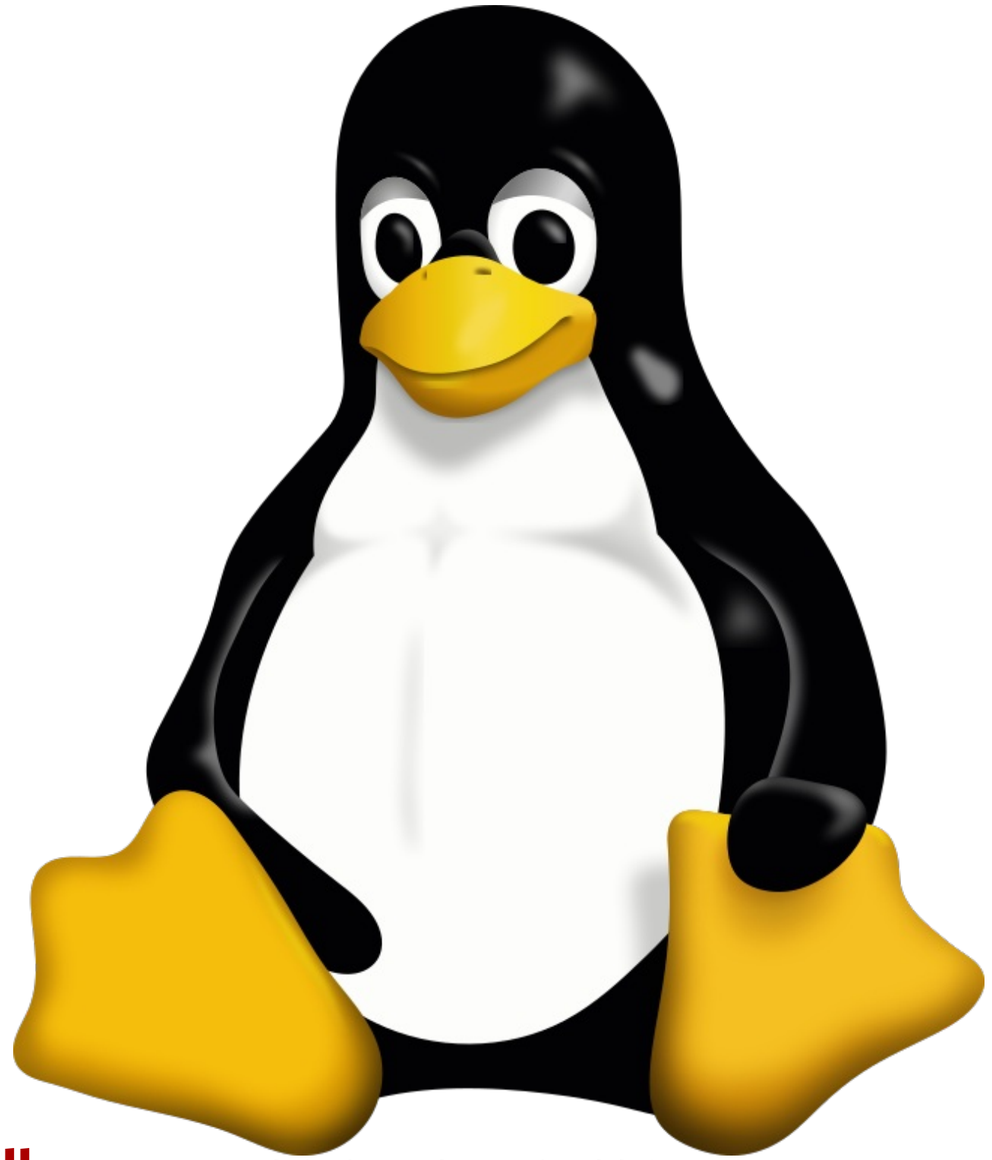
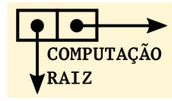
Sistemas Operacionais: arquivos e diretórios



lewing@isc.tamu.edu Larry Ewing and The GIMP, CC0,
Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Tux.svg)

Prof. Abrantes Araújo Silva Filho

Por que aprender a CLI?



Imagine a seguinte situação: você quer copiar diversos arquivos de código fonte em C de um diretório para outro, mas só quer copiar os arquivos que não existem no diretório de destino ou que são mais novos do que as versões no diretório de destino.

Diretório Fonte

```
arquivo1.c
arquivo2.c
arquivo3.c
arquivo4.c
arquivo5.c
arquivo6.c
...
```

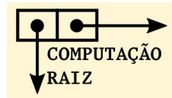
Os arquivos 3, 6, 9 e diversos outros são mais novos do que no diretório fonte do que no diretório de destino. Há centenas de arquivos.

Diretório Destino

```
arquivo2.c
arquivo3.c
arquivo5.c
arquivo6.c
arquivo7.c
arquivo9.c
...
```

Como você faria isso na GUI?

Por que aprender a CLI?



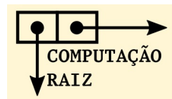
E como você faria isso na CLI?

```
cp -u *.c destino
```

A GUI é útil para realizar tarefas simples de manipulação de arquivos e diretórios. Para tarefas complexas e mais sofisticadas, não há nada que se compare com o poder e flexibilidade da CLI (todos os comandos têm dezenas de opções que simplesmente não existem na GUI).

Quer ser produtivo de verdade ao usar um sistema operacional, aprenda - DOMINE - o uso da CLI.

O que vamos aprender nesta aula?



A manipulação básica de arquivos e diretórios. Estudaremos os comandos:

- **`mkdir`**
- **`cp`**
- **`mv`**
- **`rm`**
- **`ln`**

Também aprenderemos sobre globbing, o uso de curingas para expandir nomes de arquivos e diretórios no shell baseando-se em um padrão de caracteres.

Quer ser mais produtivo no uso de um sistema operacional com a CLI? DOMINE COMPLETAMENTE o uso dos mecanismos de expansão do shell (incluindo o globbing) e expressões regulares.

Globbing

Grande parte do poder da CLI para trabalhar com arquivos e diretórios é a capacidade de utilizarmos padrões de caracteres para "casar" texto através de curingas (wildcards). Isso é chamado de globbing.

O globbing é utilizado em shells UNIX/Linux para realizar uma parte do processo de expansão do shell, especificamente o processo de expansão de nomes de arquivos. O globbing não é usado para o conteúdo de arquivos, apenas para expandir os nomes. Por exemplo: como listar todos os arquivos que terminam com a extensão `.txt`?

```
ls -l *.txt
```

Note que o `*` é um curinga que expande os nomes de arquivos terminados em `.txt`.

O globbing expande um padrão com curingas para uma lista de arquivos ou diretórios que correspndem a esse padrão.

O processo de expansão do shell, que inclui o globbing, é bem mais complexo. Veremos em aulas futuras.

Globbing

Padrão	Nome/Categoria	Descrição (com exemplo)
*	Asterisco	Casa com **qualquer seqüência** de caracteres (inclusive vazio), **sem atravessar** /. Exemplo: listar todos os arquivos que terminam com o padrão .txt: `ls *.txt`
?	Interrogação	Casa com **um único caractere**, exceto /. Exemplo: listar todos os arquivos que têm exatamente um caractere antes de .c: `ls ?.c`
[abc]	Conjunto ou Lista	Casa com UM caractere da lista. Exemplo: listar arquivos terminados em .c OU em .h: `ls *.[ch]`

Globbing

Padrão	Nome/Categoria	Descrição (com exemplo)

[a-z]	Intervalo	Casa com UM caractere dentro do **intervalo especificado**. Exemplo: listar arq0.txt até arq9.txt:
		`ls arq[0-9].txt`
		Pode-se especificar mais de um intervalo ao mesmo tempo:
		`ls arq[0-9A-Z].txt`

[!abc]	Negação	Casa com UM caractere **não listado** (no Bash usa-se !; no POSIX ^). Exemplo: listar arquivos que não começam por a: `ls [!a]*.txt`
[!a-z]	Negação de intervalo	Casa com UM caractere **não listado no intervalo**. Pode-se negar mais de um intervalo ao mesmo tempo: `ls [!0-9a-z]*.txt`

Globbing

Padrão	Nome/Categoria	Descrição (com exemplo)
`[[:class:]]`	Classe POSIX	Casa com um caractere **da classes POSIX** indicada (melhor que intervalos). Exemplo: listar arquivos que terminam com qualquer dígito: `ls *[[:digit:]]`
`[![:class:]]`	Negação de Classe POSIX	Casa com um caractere que não está na classe POSIX indicada. Exemplo: listar arquivos que não começam por letras minúsculas: `ls [![:lower:]]*`

Globbing

Classe	Equivalente	Significado
[:alnum:]	[0-9A-Za-z]	Alfanumérico
[:alpha:]	[A-Za-z]	Alfabético (somente letras ASCII)
[:digit:]	[0-9]	Apenas números
[:lower:]	[a-z]	Alfabéticos minúsculos
[:upper:]	[A-Z]	Alfabéticos maiúsculos
[:blank:]	espaço, \t	Espaço em branco restrito (espaço/tab)
[:space:]	espaço, \t, \n, \r, \v, \f, ...	Qualquer caractere de espaçamento em branco amplo
[:punct:]	pontuação	Sinais de pontuação

Globbing

Classe	Equivalente	Significado
[:cntrl:]	Caracteres de controle	Caracteres de controle ASCII (0-31, 127), não imprimíveis
[:graph:]	[!-~]	Qualquer caractere imprimível, exceto o espaço: [:alnum:] + [:punct:]
[:print:]	[-~]	Qualquer caractere imprimível, incluindo o espaço: [:graph:] + [:blank:]
[:xdigit:]	[0-9A-Fa-f]	Dígito hexadecimal
[:ascii:]		Qualquer caractere de ASCII 7 bits

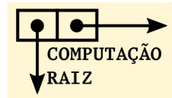
Globbing

O uso de classes POSIX é altamente recomendado pois:

- **Aumenta a portabilidade:**
  Intervalos como [A-Za-z] podem depender da ordem dos caracteres na tabela de codificação do sistema (ASCII ou UTF-8) e, em algumas situações, o intervalo pode não se comportar como esperado ou incluir caracteres acentuados.

- **Internacionalização:**
  As classes POSIX levam em conta a localização (locale) do sistema, e podem identificar corretamente (se o sistema estiver configurado com o locale correto), caracteres acentuados ou especiais.

Globbing



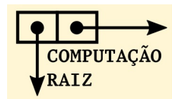
Além do globbing básico que acabamos de ver, alguns shells disponibilizam um globbing estendido, com outros curingas e funcionalidades. Essas funcionalidades podem estar ligadas ou desligadas no seu shell. Para verificar, no bash use o comando:

```
shopt extglob
```

O comando `shopt` serve para visualizar, ligar ou desligar algumas funcionalidades do bash. A opção `extglob` habilita ou desliga o globbing estendido.

Para verificar: `shopt extglob`
Para habilitar: `shopt -s extglob`
Para desligar: `shopt -u extglob`

Globbing



Ao habilitarmos o globbing estendido, as seguintes funcionalidades estarão disponíveis:

Padrão Estendido	Descrição
?(padrão)	Corresponde a 0 ou 1 ocorrência do padrão
*(padrão)	Corresponde a 0 ou mais ocorrências do padrão
+(padrão)	Corresponde a 1 ou mais ocorrência do padrão
@(padrão)	Corresponde a extamente 1 ocorrência do padrão
!(padrão)	Corresponde a qualquer coisa, exceto o padrão

O globbing estendido ajuda ao executarmos comandos mais sofisticados. O que os comandos abaixo estão fazendo?

```
ls !(*.txt)
```

```
ls +([![:lower:]])
```

Globbing

O Bash fornece outras opções para o globbing, além do globbing estendido. Algumas:

```
dotglob          off
extglob          on
failglob         off
globasciiranges  on
globskipdots     on
globstar         off
nocaseglob       off
nullglob         off
```

`extglob` globbing estendido

`globstar` super-curinga recursivo **

`nocaseglob` torna o globbing insensível a maiúsculas/minúsculas

`dotglob` faz os curingas casarem arquivos que começam com ponto

`nullglob` controla o que ocorre quando o padrão não encontra nenhuma correspondência, faz o padrão expandir para nada (útil em scripts)

`failglob` alternativa ao nullglob, faz o padrão gerar um erro e abortar

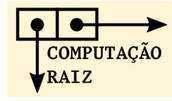
Globbing x Processo de Expansão

O mecanismo de globbing (a expansão de nomes) faz parte de um processo mais geral de expansão do shell. Isso porque o shell não executa os comandos exatamente conforme são digitados. O shell realiza uma série de etapas de processamento para expandir e substituir nomes, curingas, variáveis e outros símbolos especiais:

1. **Expansão de Braces**
2. **Expansão de Til**
3. **Expanão de:**
    - 3.1. **Variáveis e parâmetros**
    - 3.2. **Aritmética de inteiros**
    - 3.3. **Comandos**
4. **Divisão em palavras (word splitting)**
5. **Expansão de nomes (globbing)**

Sim, até agora só vimos isso aqui...

Criar diretórios



O comando `mkdir` é usado para criar diretórios. Seu uso básico é o seguinte:

```
mkdir dir
mkdir dir1 dir2 dir3
```

Para criar os diretórios pai se não existirem (não causa erro se os diretórios já existirem, nem afeta o conteúdo):

```
mkdir -p dir1/dir2/dir3
```

Copiar arquivos e diretórios

O comando `cp` faz a cópia de arquivos ou diretórios. Ele pode ser utilizado de diferentes maneiras. O uso básico é o seguinte:

Para copiar SOURCE para DEST:

```
cp [opcoes] SOURCE DEST
```

Para copiar múltiplos SOURCEs para um DIRECTORY:

```
cp [opcoes] SOURCE1 SOURCE2... DIRECTORY
```

O comando `cp` tem muitas opções importantes que precisamos aprender para, por exemplo, copiar um diretório com todo seu conteúdo.

As principais opções estão listadas a seguir.

Copiar arquivos e diretórios

Opções importantes do comando `cp`:

-a, --archive
copia os arquivos, diretórios e todos os atributos, de modo recursivo

-f, --force
se o arquivo de destino não puder ser aberto, apaga e tenta de novo (sem efeito se usar -n)

-i, --interactive
pergunta antes de sobrescrever alguma coisa

-n, --no-clobber
não sobrescreve arquivos existentes e não falha

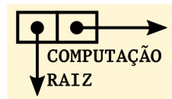
-R, -r, --recursive
copia diretórios e arquivos de forma recursiva

--update[=UPDATE]
controla que arquivos existentes serão atualizados; UPDATE={all,none,older(default)}

-u equivalente à --update[=older]

-v, --verbose
exibe mensagens informativas

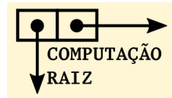
Copiar arquivos e diretórios



Exemplos do comando `cp`:

`cp file1 file2`	se file2 existir, é sobrescrita
`cp -i file1 file2`	se file2 existir, pergunta
`cp file1 file2 dir1`	dir1 deve existir
`cp dir1/* dir2`	conteúdo de dir1 para dir2, que deve existir
`cp -r dir1 dir2`	copia dir1 para dir2 (se não existir é criado)
`cp -a dir1 dir2`	copia dir1 para dir2 (se não existir é criado)

Mover e renomear arquivos e diretórios



O comando `mv` faz duas coisas básicas: move arquivos e diretórios de um lugar para outro, e troca o nome de arquivos e diretórios.

Para renomear SOURCE para DEST:

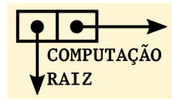
```
mv [opcoes] SOURCE DEST
```

Para mover SOURCE(s) para um DIRECTORY:

```
mv [opcoes] SOURCE1 SOURCE2... DIRECTORY
```

O comando `mv` também tem muitas opções importantes. Algumas serão listadas a seguir.

Mover e renomear arquivos e diretórios



Opções importantes do comando `mv`:

`-f, --force`
não pergunta antes de sobrescrever alguma coisa

`-i, --interactive`
pergunta antes de sobrescrever alguma coisa

`-n, --no-clobber`
não sobrescreve arquivos existentes e não falha

`-R, -r, --recursive`
copia diretórios e arquivos de forma recursiva

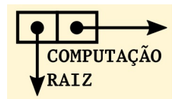
`--update[=UPDATE]`
controla que arquivos existentes serão substituídos;
UPDATE={all,none,older(default)}

`-u` equivalente à --update[=older]

`-v, --verbose`
exibe mensagens informativas

Se usar -f, -i ou -n ao mesmo tempo, só a última opção especificada é que tem efeito.

Mover e renomear arquivos e diretórios



Exemplos do comando `mv`:

`mv file1 file2`	renomeia file1 para file2; se file2 existir, é sobrescrito sem aviso
`mv -i file1 file2`	idem; se file2 existir, pergunta
`mv file1 file2 dir1`	move file1 e file2 para dentro de dir1; dir1 deve existir
`mv dir1 dir2`	se dir2 não existir, renomeia dir1 para dir2; se dir2 existir, move dir1 para dentro de dir2

Apagar diretórios vazios

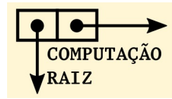
O comando `rmdir` apaga (remove) diretórios vazios. Se o diretório não estiver completamente vazio o comando termina com uma mensagem de erro e não apaga o diretório.

Não é muito usado hoje, mas ainda serve em alguns casos.

Para apagar o diretório teste (que está vazio):

```
rmdir teste
```

Apagar arquivos e diretórios (vazios ou não)



O comando `rm` apaga arquivos e diretórios (que podem estar vazios ou não). Por padrão ele não apaga diretórios, mas isso pode ser feito utilizando-se a opção de recursividade.

Para apagar um arquivo:

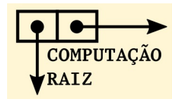
```
rm [opcoes] FILE
```

Para apagar um diretório:

```
rm -r DIRECTORY
```

O comando `rm` também tem muitas opções importantes. Algumas serão listadas a seguir.

Apagar arquivos e diretórios (vazios ou não)



Opções importantes do comando `rm`:

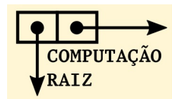
`-f, --force`
ignorar arquivos e argumentos inexistentes, não perguntar

`-i`
perguntar antes de cada remoção

`-R, -r, --recursive`
remove diretórios e seus conteúdos, recursivamente

`-v, --verbose`
exibe mensagens informativas

Apagar arquivos e diretórios (vazios ou não)



Exemplos do comando `rm`:

`rm file1 file2`	**apaga file2 e file2; não pergunta**
`rm -i file1 file2`	**idem; pergunta antes de cada arquivo**
`rm -r file1 dir1`	**apaga file1 e dir1 (incluindo conteúdo)**
`rm -fr file1 dir1`	**apaga file1 e dir1 (includindo conteúdo); se file1 e/ou dir1 não existirem, não causa erro e continua o processo de remoção**

Apagar arquivos e diretórios que começam com hífen

O que ocorre se você tentar apagar um arquivo cuja primeira letra seja um hífen? (armadilha para iniciantes)

```
abrantesasf@server01:~$ ls
-meuArquivo.txt
```

Um erro!

```
abrantesasf@server01:~$ rm -meuArquivo.txt
rm: invalid option -- 'm'
Try 'rm ./-meuArquivo.txt' to remove the file '-meuArquivo.txt'.
Try 'rm --help' for more information.
```

Apagar arquivos e diretórios que começam com hífen

Há dois modos de apagar um arquivo ou diretório que começa com um hífen:

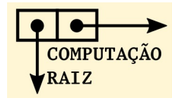
a) Usar a opção "--", que informa ao comando que não há mais nenhuma opção posterior:

```
rm -- -meuArquivo.txt
```

b) Usar o caminho relativo do diretório atual:

```
rm ./-meuArquivo.txt
```

Arquivos, diretórios e inodes



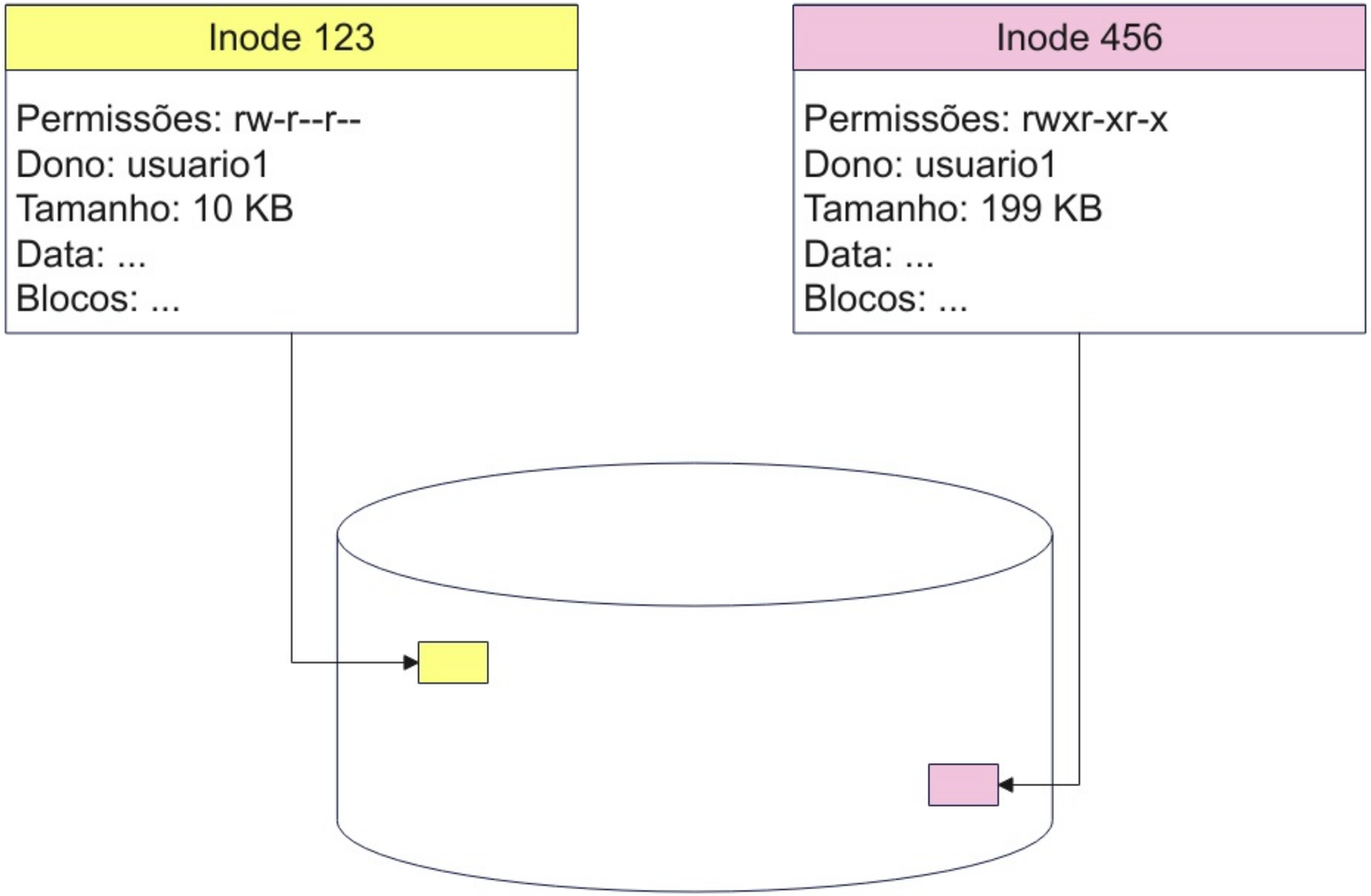
Para entendermos melhor os conceitos de arquivos, diretórios, hard links e soft links, temos que conhecer melhor a estrutura técnica de como os arquivos são armazenados no UNIX e derivados.

INODE (index node):

Cada arquivo e diretório no sistema possui um número de índice único que o identifica para o sistema operacional. É a referência principal do sistema para esse arquivo, e contém todas as informações (metadados) sobre o arquivo, mas não contém o nome do arquivo. O inode é uma estrutura que contém:

- **permissões de acesso (quem pode ler, escrever executar)**
- **dono (usuário e grupo)**
- **tamanho**
- **datas (criação, modificação e último acesso)**
- **localização dos blocos de dados do arquivo no disco**
- **NÃO CONTÉM o nome do arquivo**

Arquivos, diretórios e inodes



Arquivos, diretórios e inodes

DIRETÓRIO:

É um tipo especial de arquivo cujo conteúdo é uma tabela de mapeamento, que lista os nomes dos arquivos e seus inodes. Funciona apenas como uma "agenda de arquivos", ou seja, os nomes de aquivos que estão "dentro" desse diretório, com os respectivos números de inodes.

/home/aluno1

Arquivo	Inode
arq1.txt	123
doom.exe	456

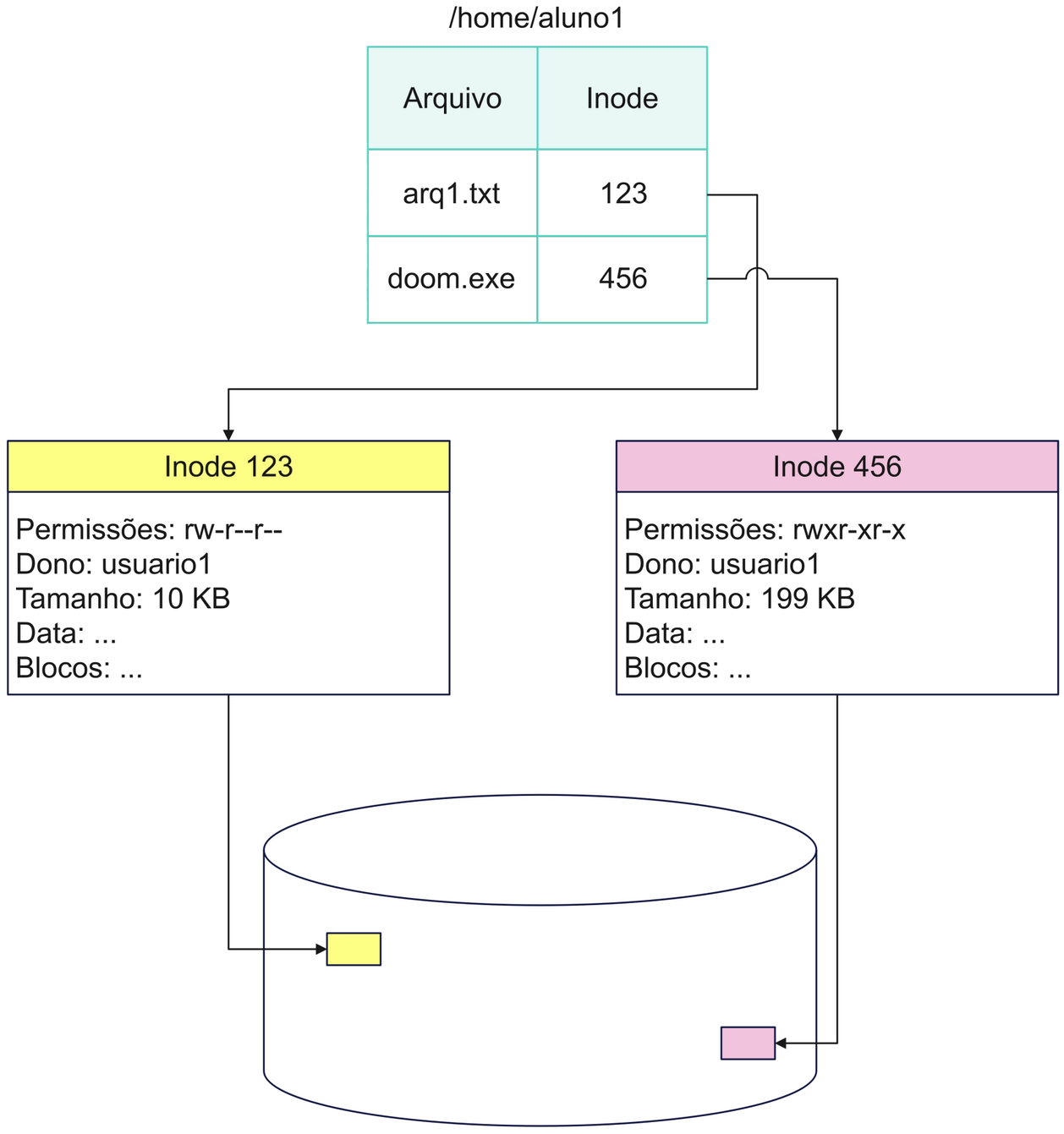
Arquivos, diretórios e inodes

Um diretório é um arquivo especial que contém uma tabela de mapeamento de nomes de arquivos para inodes;

Um inode é uma estrutura que armazena todos os metadados dos arquivos, incluindo os blocos no disco onde o arquivo está.

Um arquivo é, na verdade, apenas um conjunto de bytes ordenados no disco.

O nome do arquivo é só um rótulo amigável.



Hard links

Por padrão, todo arquivo no UNIX (e derivados) tem um hard link único que fornece o nome do arquivo, ou seja, um hard link é um nome de arquivo na tabela do diretório que aponta para um inode.

Ao criarmos hard links adicionais estamos criando nomes adicionais para o mesmo arquivo, permitindo acessar os mesmos dados a partir de múltiplos caminhos dentro do mesmo sistema de arquivos.

Estamos, de fato, criando mais nomes para o mesmo inode!

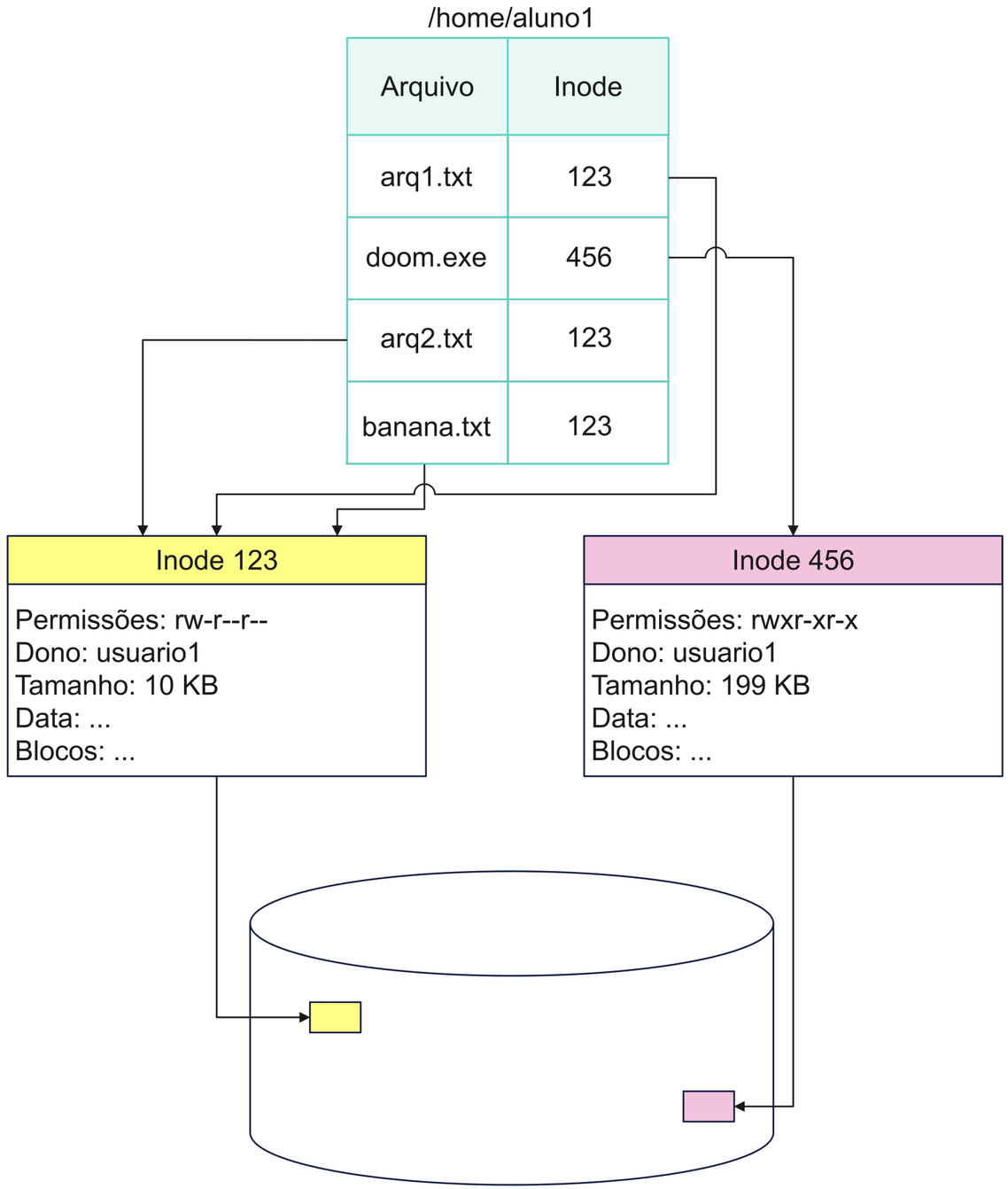
Hard links

O nome inicial do arquivo apontado pelo inode 123 é "arq1.txt". Esse é o 1º hard link para o arquivo.

Depois nós criamos mais 2 nomes para esse arquivo: "arq2.txt" e "banana.txt". Esses são o 2º e o 3º hard links para o arquivo apontado pelo inode 123.

Usamos o comando `ln` para criar hard links (`ln TARGET LINK`)

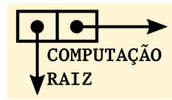
```
ln arq1.txt arq2.txt
ln arq1.txt banana.txt
```



Hard links

```
abrantesasf@server01:~$ ls -l
total 4
-rw-rw-r-- 1 abrantesasf abrantesasf 9 Aug 26 06:54 arq1.txt
abrantesasf@server01:~$ ln arq1.txt arq2.txt
abrantesasf@server01:~$ ls -l
total 8
-rw-rw-r-- 2 abrantesasf abrantesasf 9 Aug 26 06:54 arq1.txt
-rw-rw-r-- 2 abrantesasf abrantesasf 9 Aug 26 06:54 arq2.txt
abrantesasf@server01:~$ ln arq1.txt banana.txt
abrantesasf@server01:~$ ls -l
total 12
-rw-rw-r-- 3 abrantesasf abrantesasf 9 Aug 26 06:54 arq1.txt
-rw-rw-r-- 3 abrantesasf abrantesasf 9 Aug 26 06:54 arq2.txt
-rw-rw-r-- 3 abrantesasf abrantesasf 9 Aug 26 06:54 banana.txt
abrantesasf@server01:~$ ls -li
total 12
655374 -rw-rw-r-- 3 abrantesasf abrantesasf 9 Aug 26 06:54 arq1.txt
655374 -rw-rw-r-- 3 abrantesasf abrantesasf 9 Aug 26 06:54 arq2.txt
655374 -rw-rw-r-- 3 abrantesasf abrantesasf 9 Aug 26 06:54 banana.txt
abrantesasf@server01:~$ _
```

Hard links



Importante:

- **Um hard link é um nome que aponta para o mesmo inode, então todos os hard links apontam para o mesmo arquivo físico (mesmo bloco no disco)**
- **Alterar o arquivo a partir de qualquer hard link, altera para todos**
- **Os hard links são indistinguíveis, não há um arquivo "original" e "cópias", todos têm o mesmo "peso" pois são o mesmo**
- **O arquivo só é apagado do disco quando todos os hard links são removidos**
- **É possível criar hard links para outros hard links**

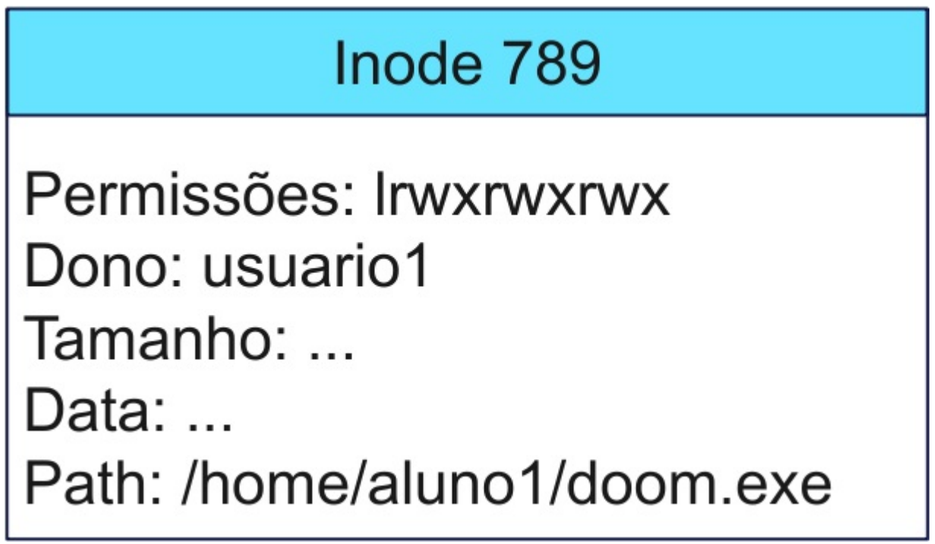
Restrições:

- **Não podem apontar para diretórios (exceto . e ..)**
- **Não podem atravessar sistemas de arquivos diferentes (pois cada sistema de arquivos tem sua própria tabela de inodes), por exemplo, em outra partição do disco**

Symbolic links

Não são nomes diferentes para o mesmo inode, são um tipo especial de arquivo que guarda o caminho para o arquivo ou diretório referenciado.

Um link simbólico possui um inode próprio que contém o path, o caminho para o arquivo. Na prática funciona como um "atalho" até o arquivo ou diretório referenciado.

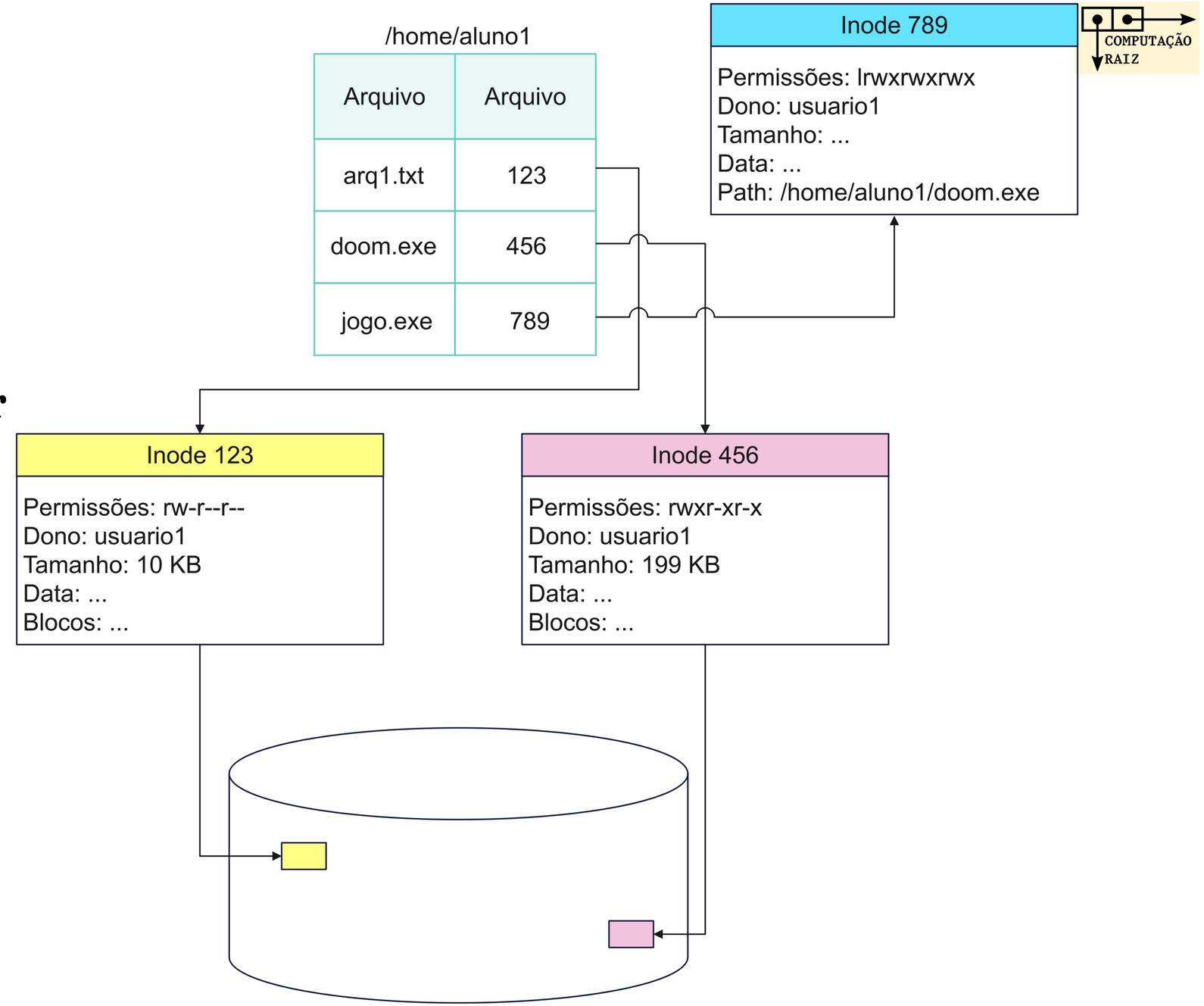


Symbolic links

Para todos os efeitos, quando você acessa um link simbólico, está acessando o arquivo referenciado.

Usamos o comando `ln` para criar hard links, com a opção `-s` (`ln -s TARGET LINK`)

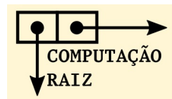
```
ln -s doom.exe jogo.exe
```



Symbolic links

```
abrantesasf@server01:~$ ls -l
total 4
-rw-rw-r-- 1 abrantesasf abrantesasf 9 Aug 26 07:33 arq1.txt
abrantesasf@server01:~$
abrantesasf@server01:~$ ln -s arq1.txt banana.txt
abrantesasf@server01:~$
abrantesasf@server01:~$ ls -l
total 4
-rw-rw-r-- 1 abrantesasf abrantesasf 9 Aug 26 07:33 arq1.txt
lrwxrwxrwx 1 abrantesasf abrantesasf 8 Aug 26 07:34 banana.txt -> arq1.txt
abrantesasf@server01:~$
abrantesasf@server01:~$ ln -s banana.txt uva.txt
abrantesasf@server01:~$
abrantesasf@server01:~$ ls -l
total 4
-rw-rw-r-- 1 abrantesasf abrantesasf  9 Aug 26 07:33 arq1.txt
lrwxrwxrwx 1 abrantesasf abrantesasf  8 Aug 26 07:34 banana.txt -> arq1.txt
lrwxrwxrwx 1 abrantesasf abrantesasf 10 Aug 26 07:34 uva.txt -> banana.txt
abrantesasf@server01:~$
```

Symbolic links



Importante:

- Ao apagarmos o link simbólico, NÃO apagamos o arquivo referenciado, apagamos apenas o link simbólico

- Se apagarmos o arquivo, o link ficará quebrado

- Pode se referir a arquivos e diretórios

- Pode ultrapassar sistemas de arquivos diferentes

- É possível criar mais de um link simbólico para o mesmo arquivo

- É possível criar links simbólicos para outros links simbólicos

Hard x Symbolic links

📊 Comparação entre Hard Links e Soft Links

Característica	Hard Link	Soft Link (Symlink)
O que é	Outro nome para o mesmo arquivo (mesmo inode).	Um arquivo especial que contém o caminho para outro arquivo/diretório.
Inode	Aponta para o **mesmo inode** do arquivo original.	Possui **inode próprio**, que aponta para o caminho do arquivo original.
Conteúdo	Compartilha exatamente os mesmos blocos de dados.	Apenas um "atalho" (caminho) até o alvo.
Efeito se o original for apagado	O arquivo continua existindo (outro hard link ainda referencia o inode).	O link fica "quebrado" (dangling link), apontando para um alvo inexistente.
Atravessa sistemas de arquivos	❌ Não pode. Precisa estar no mesmo sistema de arquivos.	✔️ Pode apontar para arquivos em outro sistema de arquivos ou partição.
Pode apontar para diretórios	❌ Normalmente não (exceto `.` e `..`).	✔️ Pode apontar para diretórios.
Identificação	`ls -li` mostra o mesmo número de inode.	`ls -li` mostra inode diferente; `ls -l` mostra seta → caminho do destino.
Velocidade de acesso	Direto (mesmo inode).	Um nível indireto (precisa resolver o caminho do alvo).
Uso típico	Garantir redundância de nomes para o mesmo arquivo; evitar p ao deletar.	Criar atalhos, apontar para arquivos/diretórios em locais diferentes.