Inf 514 – Linux
parte 1

prof. Fernando Vanini
(vanini@ic.unicamp.br)

Porque Linux ?

- Sistema operacional de código ‘aberto’
- Estrutura simples e coerente
- Baseado nos conceitos do UNIX
- Disponível em várias plataformas de hardware

- O UNIX influenciou a maioria dos sistemas operacionais posteriores a ele, incluindo o Linux
  - O macOS, sistema operacional da Apple para o macBook é baseado no BSD, que é uma implementação do UNIX
  - O Android, usado na maioria dos smartphones atuais é baseado no Linux.
  - O ChromeOs, usado no laptop ChromeBook também é baseado no Linux.

O Unix

- O projeto do UNIX segue um conjunto de princípios que tem se mostrado efetivos em qualquer projeto.
- Esses princípios se baseiam na decomposição de qualquer sistema em partes simples que por sua vez são combinadas através de um conjunto de mecanismos de composição também simples.

O Unix

- O UNIX definiu um conjunto desses mecanismos de composição que acabou se tornando base para um padrão adotado por praticamente qualquer sistema operacional desenvolvido depois dele.

O Unix

- A flexibilidade oferecida pelos serviços do UNIX possibilitou a criação de novos conceitos em várias áreas da computação.
- A área de redes em geral foi muito beneficiada.
- Por exemplo, o protocolo TCP/IP foi desenvolvido por um grupo que trabalhava na implementação de uma das primeiras versões *free* do UNIX (free BSD).

Linux e Unix

- A AT&T, detentora dos direitos do UNIX, foi proibida pela legislação americana de comercializar produtos de informática, incluindo o UNIX.
- Em função do sucesso do UNIX em instituições de pesquisa, diversos fabricantes criaram suas versões proprietárias do UNIX.
- O grupo Berkely Software Distribution, da Universidade de Berkeley desenvolveu o free BSD, uma versão 'open source' do UNIX.
- Posteriormente o estudante Linus Torvalds desenvolveu, por iniciativa própria, o Linux, também uma versão de código aberto do UNIX.
- O Linux acabou virando uma espécie de febre entre os adeptos de código aberto e diversas distribuições Linux foram desenvolvidas, com porte para variadas plataformas de hardware.
- A Microsoft, através de uma parceria com distribuidores de Linux, desenvolveu o WSL, Windows subsystem for Linux, que permite a instalação e execução de aplicativos Linux sob o Windows.

Sistema Operacional

- Um Sistema Operacional é um programa ou conjunto de programas, responsável por administrar os recursos oferecidos pelo hardware e por oferecer aos usuários um conjunto de serviços.
- A forma mais comum pela qual o sistema operacional oferece seus serviços aos usuários é através da execução de *programas* aplicativos.
- Normalmente o sistema operacional estende o hardware através da criação de conceitos como arquivos, janelas, botões, etc.

Sistema Operacional

- Os recursos normalmente gerenciados pelo sistema operacional são os seguintes:
  - Memória
  - Tempo do processador
  - Periféricos (hd, teclado, mouse, interface de rede, etc.)

O sistema operacional oferece aos programas de aplicação uma 'camada de abstração', criando alguns conceitos próprios, que não são oferecidos pelo 'hardware puro'. Exemplos:

- Arquivos
- Programas

Multiprogramação

- Um sistema operacional é dito *multiprogramado* se o mesmo permitir que mais de um programa seja executado "simultaneamente".
- A "execução simultânea" na verdade é uma execução intercalada de trechos de cada um dos programas.
- Como isso ocorre de forma muito rápida, o usuário tem a impressão que os programas são executados simultaneamente.

Programas e processos

- Um *processo* corresponde a um programa em execução.
- Normalmente o termo *programa* é utilizado para definir o código do mesmo e o termo *processo* para se referir à ativação do mesmo, composta por código, dados, arquivos e outros recursos necessários à execução.
- Num sistema multiprogramado, um mesmo programa pode dar origem a vários processos.

Objetivos iniciais do UNIX

- O UNIX foi projetado para ser um sistema operacional de *timesharing*, para ser utilizado principalmente por programadores, trabalhando em projetos de forma cooperativa.
- O sistema oferece recursos que permitem às pessoas trabalhar em conjunto, compartilhando informações de forma controlada.
- Esse modelo de trabalho é bem diferente daquele no qual um usuário sem muita experiência trabalha sozinho num computador pessoal, usando um processador de textos.

Objetivos iniciais do UNIX

- Tendo sido desenvolvido "de programadores para programadores", o UNIX oferece os recursos numa forma que em geral agrada à maioria dos programadores e usuários experientes.
- Os comandos e utilitários, por exemplo, têm uma interface simples e direta, o que nem sempre é intuitivo para os iniciantes.

Estrutura geral do UNIX/Linux

- O UNIX/Linux pode ser visto como uma estrutura formada por várias camadas superpostas:
  - Hardware - máquina sobre a qual o UNIX é executado.
  - Sistema Operacional (kernel) – gerencia o hardware e oferece um conjunto de serviços disponibilizados através de um conjunto de *system calls*.

Estrutura geral do UNIX/Linux

- O sistema operacional tem a função de gerenciar o hardware e oferecer um conjunto de *system calls* a todos os programas que executam sob seu controle.
- O sistema operacional é responsável também por estender o hardware através da criação de serviços relacionados com arquivos, processos, dispositivos lógicos, etc.

Bibliotecas Padrão

- As chamadas ao sistema são feitas em baixo nível, através de valores colocados em registradores e de instruções especiais para chamada ao sistema.
- As bibliotecas padrão oferecem uma interface através da qual as chamadas ao sistema podem ser feitas através de uma linguagem de alto nível, como por exemplo C.

Programas de Aplicação

- Os programas de aplicação e utilitários em geral utilizam os serviços do sistema operacional através das funções definidas nas bibliotecas padrão.
- São os responsáveis por interagir diretamente com o usuário.

Estrutura Geral

Arquivos e Diretórios

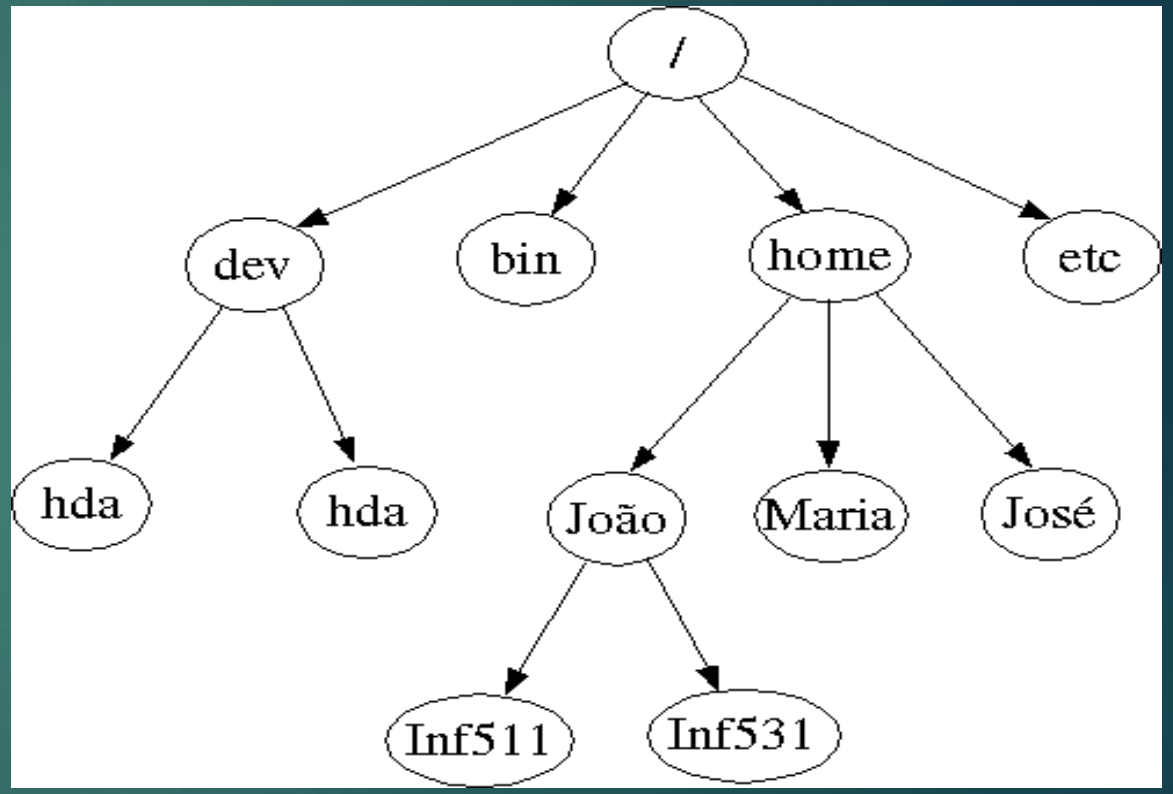
- Além dos bytes que constituem o conteúdo de um arquivo, o UNIX mantém associado ao mesmo
  - a indicação do usuário que é o seu "dono" (*owner*)
  - o *grupo* ao qual este usuário pertence
  - um conjunto de bits de proteção indicando o que pode ser feito com esse arquivo e por quem.

Arquivos e Diretórios

- Um diretório é um arquivo que contém informações a respeito de um conjunto de arquivos.
- Essas informações consistem basicamente do nome do arquivo e localização do mesmo.
- Um arquivo "contido" num diretório pode ser por sua vez um diretório, permitindo que se organize de forma hierárquica os arquivos do sistema.

Estrutura de Diretórios

O UNIX define uma estrutura de diretórios, usada como base para a localização dos principais arquivos do sistema.



Estrutura de Diretórios

- /bin –
  - contém arquivos com programas executáveis (comandos)
- /dev
  - *arquivos especiais* associados a despositivos (*devices*)
- /etc
  - arquivos de administração e configuração
- /home
  - contém os diretórios de usuários
- /lib
  - bibliotecas padrão
- /sbin
  - comandos gerais de administração do sistema
- /tmp
  - arquivos temporários
- /usr
  - contém vários diretórios (a notação "dir/arq" indica o arquivo "arq" no diretório "dir") :

Estrutura de Diretórios

- /usr/X11 arquivos de sistema do gerenciador de janelas X Windows
- /usr/adm arquivos de dados de administração do sistema
- /usr/bin comandos executáveis (adicionais)
- /usr/lib bibliotecas e pacotes de configuração
- /usr/local/bin comandos adicionais locais
- /usr/local/lib bibliotecas adicionais locais
- /usr/local/src arquivos fontes dos comandos locais
- /usr/man manuais online (*man pages*)
- /usr/src/linux arquivos fonte do kernel do Linux
- /var área de *overflow* para alguns arquivos

Processos

- Num sistema multiprogramado, o conceito de processo é fundamental. O UNIX/Linux oferece um conjunto de *system calls* orientado à criação e comunicação entre processos pelos programas de aplicação.
- Um processo corresponde à *ativação de um programa*. Esse "programa" pode ser um programa de aplicação desenvolvido pelo próprio usuário ou um dos utilitários do próprio UNIX.

Processos

- Através das *system calls* um programa de usuário pode criar vários processos que executam "simultaneamente", podendo se comunicar durante a sua execução.
- Esse tipo de facilidade representa um recurso arquitetural poderoso na construção de aplicações (a implementação do protocolo TCP/IP, por exemplo, utiliza esses recursos).

Usuários e Controle de Acesso

- Um sistema operacional multi-usuário deve garantir a integridade das informações associadas a cada usuário.
- Para ter acesso aos serviços oferecidos pelo UNIX/Linux, o usuário deve se identificar para o sistema, fornecendo o seu nome e senha (*login*).
- A partir de então ele só tem acesso aos recursos que lhe forem designados.

Shell: a Interface do usuário

- O UNIX foi concebido numa época em que os sistemas de *time sharing* ainda eram novidade.
- Nessa época, o acesso interativo era feito através de terminais "burros" (sem capacidade de processamento local), que operavam unicamente em modo texto.
- A interface de comandos original do UNIX, denominada *shell* foi projetada nesse contexto e é portanto natural que seja orientada a texto ou "linha de comando".

Shell: a Interface do usuário

- A *shell* é um programa de aplicação como outro qualquer, que utiliza as mesmas *system calls* oferecidas pelo kernel.
- Ela executa em modo usuário e eventualmente pode ser substituída.
- A partir da disseminação do UNIX, outras versões da *shell* foram desenvolvidas (*bourne shell, korn shell, C shell,* , etc...).
- Atualmente a versão mais usada em instalações Linux é a *bash* ("bourne again shell") que é uma evolução da *bourne shell*.

Shell: a Interface do usuário

- Ao executar a shell indica ao usuário que está *pronta* para receber um comando através de um string indicando "prompt".
- Esse string depende da instalação mas é configurável.
- Os *comandos* executados pela shell são na verdade programas de aplicação que executam em modo usuário.

Entrada e saída padrão

- A shell durante a sua execução utiliza dois dispositivos para a comunicação com o usuário:
  - Entrada Padrão - dispositivo a partir do qual a shell obtem os comandos a serem executados.
  - Saída Padrão - dispositivo no qual a shell escreve os resultados dos comandos.

Entrada e saída padrão

- Normalmente a entrada padrão é associada ao teclado e a saída padrão é associada à uma área na tela onde a shell escreve os resultados da execução dos comandos.
- Essa associação é estabelecida na ativação da shell.
- É possível associar esses dispositivos lógicos a qualquer outro dispositivo físico que seja compatível com as operações de leitura e escrita de caracteres.

Entrada e saída padrão

- Tanto a entrada padrão como a saída padrão podem ser associadas, por exemplo, a arquivos.
- Um comando, ao ser executado sob controle da shell, irá utilizar os mesmos dispositivos de entrada e saída padrão usados pela mesma.
- A sintaxe da shell permite que esses dispositivos sejam redirecionados.

Filtros

- Um *filtro* é um programa que lê dados pela entrada padrão e escreve os resultados pela saída padrão.
- A shell do UNIX permite que programas do tipo filtro sejam encadeados através de *pipes*, que serão discutidas mais adiante.
- Exemplo:
  - O comando *sort* ordena uma sequência de linhas de texto lidas da entrada padrão e escreve os dados ordenados na saída padrão:

```
sort
```

Redirecionamento de entrada e saída

- Um programa (ou *comando*) pode ter a sua entrada padrão redirecionada para um arquivo.
- Isso é feito através do caracter "<" seguida do nome do arquivo.
- De forma análoga, a saída padrão também pode ser redirecionada através do uso do caracter ">" seguido do nome do arquivo.

Redirecionamento de entrada e saída

Exemplos:

```
sort  <lista
sort  <lista >lista_ordenada
```

O diretório corrente

- Ao criar uma sessão, durante a qual o usuário utiliza a shell, esta se refere sempre a um determinado diretório, chamado *diretório corrente* ou *diretório atual*.
- Através do comando **`pwd`** ('print working directory') é possível identificar o diretório corrente.
- Através do comando cd ('change directory') é possível alterar o diretório corrente.

O diretório corrente

Exemplos:

`cd /`	muda para o diretório raiz
`cd /etc`	muda para o subdiretório "etc" do diretório raiz
`cd /etc/lib`	muda para o subdiretório “etc/lib” do diretório raiz
`cd tmp`	muda para o subdiretório "tmp" do diretório corrente

O comando ls

O comando ls lista o conteúdo de um diretório. Sua forma geral é a seguinte:

```
ls [opções ] [arquivos]... [arquivos]
```

O resultado é escrito na saída padrão

O comando ls

- Exemplos:

```
ls
ls -ld/etc
ls *.c
ls -la >dir
ls -a /etc
ls -R /etc
```

  - "/" se refere à *raiz* da árvore de diretórios.
  - As opções "l" e "d" indicam respectivamente *formato longo* e *diretório*. Ou seja, os diretórios contidos em /tmp serão listados em *formato longo*.
  - A opção "a" indica *todos* (*all*).
  - A opção "R" indica *Recursivamente*.

As referências “.” e “..”

- Ao listar o conteúdo de um diretório (através de p. ex. "ls -a"), aparecem na lista, como arquivos, os nomes "." e "..".
- Esses nomes são referências ao diretório corrente (".") e ao diretório que contém o diretório corrente.
- Por exemplo, se quizermos listar o conteúdo do diretório que contém o diretório corrente, podemos fazer o seguinte:
  - `ls ..`
- Caso o diretório corrente seja o *diretório raiz*, ".." se refere ao próprio diretório.

As referências “.” e “..”

- As referências "." e ".." podem ser usadas com outros comandos.
- Exemplos:

```
cd ..
cd ../tmp ls -la ../tmp
sort <../lista >../listaOrd
```

O comando cat

- O comando *cat* con*cat*ena o conteúdo da entrada padrão com o conteúdo da saída padrão.
- Este comando é bastante usado para apresentar na tela o conteúdo de um arquivo texto.
- Exemplos:

  `cat < lista` mostra na saída padrão o conteúdo do arquivo lista

  `cat > novaLista` copia os dados lidos da entrada padrão (encerrados por [control-d]) para o arquivo novaLista

Cópia de Arquivos

`cp arq1 arq2` copia o conteúdo do arquivo arq1 para o arquivo arq2. Se arq2 já existir nesse diretório, seu conteúdo será copiado sobre o mesmo. Se arq2 não existir ele será criado.

`cp arq1 arq2 arq3 dirX` copia os arquivos arq1, arq2, arq3 para o diretório dirX. Se os arquivos dirX/arq1, dirX/arq2, dirX/arq3 não existirem, eles serão criados. Caso já existam, o conteúdo deles será substituído pelo conteúdo dos arquivos sendo copiados.

O comando mv

`mv arq1 arq2`

muda o nome do arquivo "arq1" para "arq2". Se o arquivo arq2 já existir nesse diretório, o seu conteúdo será substituído pelo novo conteúdo.

`mv arq1 arq2 arq3 dir`

transfere os arquivos arq1 arq2 arq3 para o diretório dir. Se nesse diretório existirem arquivos com esses nomes, o seu conteúdo será substituído.

Outros comandos relacionados a arquivos

`less arq`

mostra na tela o conteúdo de um arquivo, permitindo que se "navegue" por ele através das teclas PgUp, PgDown, etc.

`file arq`

mostra na saída padrão tipo do conteúdo.

`locate arq`

procura recursivamente por um arquivo, indicando o diretório onde o mesmo se situa.

Curingas

- Em qualquer comando da shell no qual se utiliza um nome de arquivo, é possível utilizar *curingas* (*wild card*).
- Isso é feito através do uso de símbolos especiais que definem um padrão.
- Ao processar o comando, a shell procura os nomes de arquivo que casam com o padrão especificado e usa esses nomes no comando.
- Um dos curingas possíveis é representado pelo caracter "*", que um padrão que "casa" com qualquer sequência de (zero ou mais) caracteres.

Curingas

Exemplos:

`*.html`

casa com qualquer nome de arquivo que termine com ".html".

`pag*`

casa com qualquer nome de arquivo que se inicie com "pag".

`p*html`

casa com qualquer nome de arquivo que se inicie com "p" e termine com "html".

Curingas

O curinga "?" define um padrão que casa com qualquer caracter (neste caso "exatamente um caracter").

Exemplos:

`pag_?.html` casa com pag_1.html, pag_2.html, pag_3.html,...

`x??config*`

Como a shell trata os comandos

- A shell é responsável por ler cada linha de comando, processá-la e executar o programa correspondente.
- Cada um dos "comandos" conhecidos pela shell é associado a um programa executável (a maior parte deles fica no diretório /bin).
- O "processamento" do comando inclui a substituição dos curingas pelos nomes dos arquivos no diretório (indicado explícita ou implicitamente) que casam com os mesmos.

Como a shell trata os comandos

- Um exemplo: ao processar o comando

```
mv *.html ./paginas
```

- a shell irá procurar os nomes de arquivo que casam com o padrão "*.html". Supondo que esses arquivos sejam pag_1.html, pag_2.html e pag_3.html o comando efetivamente executado pela shell será

```
mv pag_1.html pag_2.html pag_3.html ./paginas
```

- Com isso, tratamento dos curingas fica restrito à shell.

Como a shell trata os comandos

- O tratamento dos curingas é restrito à shell.
- Os comandos não precisam tomar conhecimento da sua existência, e isso simplifica a implementação dos mesmos.
- Essa abordagem é consistente com um dos princípios de projeto do UNIX: "fazer uma coisa de cada vez, fazendo-a bem feita".
- O conjunto de "comandos conhecidos" pela shell na verdade corresponde ao conjunto de programas executáveis disponíveis (os termos *comandos* e *programas executáveis* são equivalentes).
- Ao processar uma linha de comando, a shell identifica o programa e em seguida procura pelo mesmo a partir de um *caminho de busca* definido num arquivo de configuração.

Como a shell trata os comandos

- Ao processar uma linha de comando, a shell identifica o programa e em seguida procura pelo mesmo a partir de um *caminho de busca* definido num arquivo de configuração.
- Ao digitar uma linha de comando o usuário pode indicar o diretório onde o mesmo se encontra da mesma forma que o faz com um arquivo (o que é natural já que o seu programa está contido num arquivo).

Como a shell trata os comandos

- Ao processar uma linha de comando, a shell identifica o programa e em seguida procura pelo mesmo a partir de um *caminho de busca* definido num arquivo de configuração.
- Ao digitar uma linha de comando o usuário pode indicar o diretório onde o mesmo se encontra da mesma forma que o faz com um arquivo (o que é natural já que o seu programa está contido num arquivo).

Como a shell trata os comandos

- Exemplos:

`ls *html`

executa o programa /bin/ls (o diretório /bin está no caminho de busca padrão.

`./prog x y`

executa o programa prog, que está no diretório corrente, com os parâmetros "x" e "y".

`/sbin/lilo -v`

executa o programa lilo, que está no diretório /sbin, com a opção "v".

Pipes

- Suponha que por alguma razão você precise de uma lista dos comandos disponíveis no diretório /bin em ordem alfabética.
- Para obter essa lista podemos fazer o seguinte:

```
ls /bin > temp
sort < temp
rm temp
```

Pipes

- o UNIX permite que se "concatene" a execução dos programas *p1* e *p2* através de um recurso denominado *pipe*, indicado no comando através do caracter "|". Usando *pipe* a nossa sequencia ficaria da seguinte forma:

```
ls /bin | sort
```

- O uso de *pipes* é compatível com as demais opções para os comandos.

Pipes

- Se quisermos que a nossa lista ordenada com os comandos em /bin seja guardada num arquivo, podemos redirecionar a saida de *sort*:

- `ls /bin | sort > listaOrd`

- A shell permite também que vários comandos sejam encadeados atraves de pipes, como por exemplo:

- `p1 | p2 | p3 | p4`