

Introdução

João Marcelo Uchôa de Alencar

Universidade Federal do Ceará - Quixadá

8 de Agosto de 2019

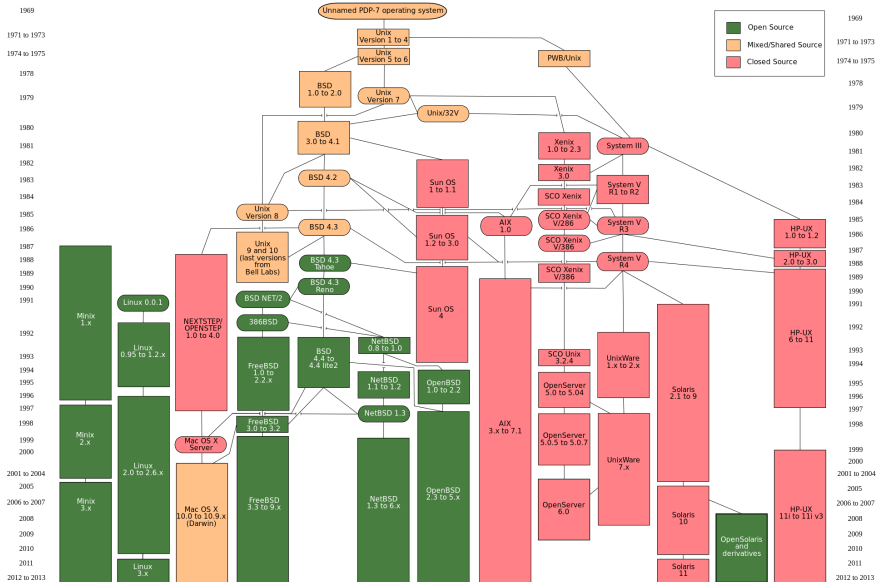
Apresentação

- ▶ Professor João Marcelo Uchôa de Alencar.
- ▶ E-mail para contato: joao.marcelo@ufc.br.
- ▶ Programa da disciplina:
`http://professor.joao.marcelo.nom.br/`.
- ▶ Avaliações:
 - ▶ 3 avaliações (Nota de Prova Prática + Atividades Semanais).
 - ▶ Seminário no final da disciplina (Nota da Apresentação + Atividades Semanais).
 - ▶ Pegar as 3 maiores notas e fazer a média.
- ▶ Proposta: não ter prova final, aqueles que tiveram acima de 5,0 estão aprovados. Caso contrário, reprovados.

História dos Sistemas UNIX

- ▶ Unix foi desenvolvido pelo *Bell Labs* logo após esse laboratório abandonar o projeto *Multics*.
- ▶ Algumas decisões de projeto foram reaproveitadas do *Multics*
 - ▶ Dispositivos como arquivos.
 - ▶ Interpretador de comandos como processo independente do *kernel*.
- ▶ O *Bourne Shell* e a linguagem *awk* foram incluídos na versão 7 do UNIX.
- ▶ Um foco forte na criação de ferramentas para processamento textual.
- ▶ Padrão POSIX.

História dos Sistemas UNIX



Princípios Para Ferramentas de *Software*

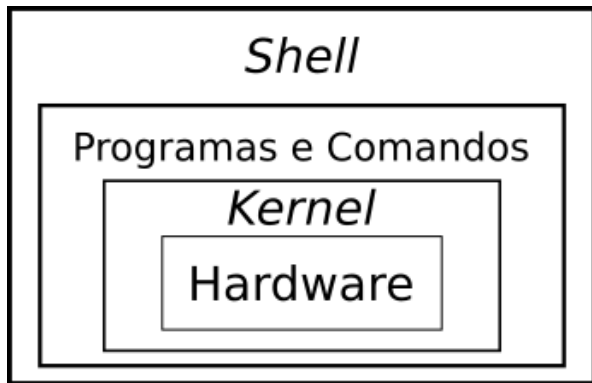
Os ambientes UNIX obedecem os seguintes princípios no desenvolvimento de suas ferramentas:

- ▶ Cada ferramenta deve oferecer apenas uma **funcionalidade** e implementá-la da melhor maneira.
- ▶ Processar **linhas de texto**, evitar dados binários.
- ▶ Aproveitar o poder das **expressões regulares**.
- ▶ Utilizar a **saída padrão** para troca de dados.
- ▶ Evitar **mensagens desnecessárias** no terminal.
- ▶ Utilizar para a saída o **mesmo formato** da entrada.
- ▶ Reutilizar programas **existentes**.
- ▶ Tentativa de **generalizar** suas ferramentas.

Linguagens de *Scripting* versus Linguagens Compiladas

- ▶ Programas escritos em linguagens compiladas são traduzidas do seu **código fonte** original para o **código objeto** da máquina alvo.
- ▶ A vantagem é que são programas eficientes, a desvantagem é que atuam em baixo nível.
- ▶ Exemplo da desvantagem: escrever em C um programa que copie todos arquivos de um diretório para outro.
- ▶ Linguagens de *script* são interpretadas, um **interpretador** lê cada linha do *script* e usa funções internas para executá-las.

O que é o *Shell*?



O que é o *Shell*?

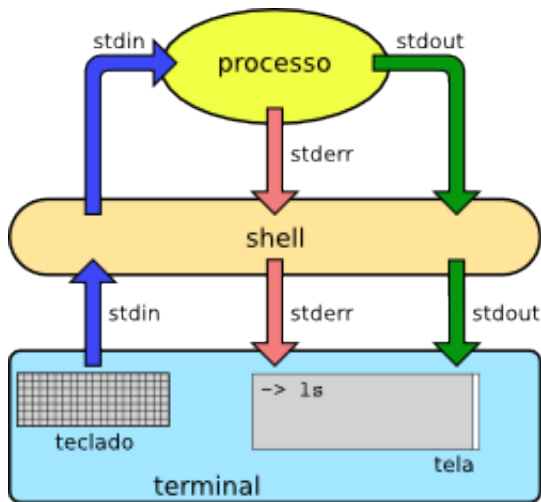
Ao nível do *Shell*:

```
$ ls
'Área de trabalho' Documentos Downloads Dropbox Imagens Modelos Música
Público Vídeos
```

Ao nível do *Kernel*:

```
execve("/usr/bin/ls", ["ls"], [/* 49 vars */]) = 0
brk(NULL) = 0x5585c31e8000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=101277, ...}) = 0
...
```


O que é *Shell*?



Fonte: http://wiki.inf.ufpr.br/maziero/doku.php?id=unix:shell_avancado

O que é o *Shell*?

- ▶ O *shell* é simplesmente o programa que lê o comando que você teclou e o converte em uma forma mais simplificada e legível ao sistema operacional (*kernel*).
- ▶ Também fornece uma **linguagem** para organizar uma sequência de comandos em *scripts*, permitindo a automação de tarefas administrativas.
- ▶ Um dos grandes diferenciais é permitir invocar programas feitos em outras linguagens sem dificuldade.
- ▶ A troca de dados entre programas é feita em formato de linhas de **texto simples**.

Tarefas do *Shell*

- ▶ Exame da linha de comandos recebida.
- ▶ Atribuição e substituição de variáveis.
- ▶ Resolução de redirecionamentos.
- ▶ Substituição de metacaracteres.
- ▶ Criação de processos filhos para executar comandos.

Principais versões do *Shell*

- ▶ *Bourne Shell* (sh)
- ▶ *Bourne Again Shell* (bash)
- ▶ *Korn Shell* (ksh)
- ▶ *C Shell* (csh)

Por que usar *Shell*?

- ▶ É mais fácil lidar com arquivos e diretórios em *Shell*, assim como iniciar ou finalizar outros programas.
- ▶ O desempenho costuma ser pior, mas o tempo de desenvolvimento é bem menor do que em linguagens compiladas.
- ▶ Simplicidade.
- ▶ Portabilidade.
- ▶ Facilidade de desenvolvimento.

Um primeiro exemplo - Contando os Usuários

```
$ who
$ who | wc -l
$ cat > usuarios.sh
who | wc -l
^D
$ chmod +x usuarios.sh
$ ./usuarios
```

Um primeiro exemplo - Contando os Usuários

- ▶ Qualquer sequência de comandos, dentro de um arquivo de texto, com permissão para execução, vira um *shell script*.
- ▶ Ao perceber que não é um arquivo compilado, o *kernel* retorna o arquivo ao *shell*, que o executa com o *shell* padrão do sistemas.
- ▶ Mas como existem vários *shells*, é boa prática especificar qual você deseja utilizar.

Um primeiro exemplo - Contando os Usuários

```
$ cat usuarios.sh  
#!/bin/bash  
who | wc -l
```

Neste contexto, o seguinte comando:

```
$ ./usuarios.sh
```

equivale a:

```
$ /bin/bash usuarios.sh
```


OFF-TOPIC: Como Usar o SSH/SCP

- ▶ O SSH é um protocolo para *login* remoto que utiliza criptografia para proteger a troca de informações.
- ▶ O comando *ssh* no Linux implementa o lado cliente do protocolo.
- ▶ O protocolo SSH também pode ser usado para transferir arquivos, não só enviar comandos.
- ▶ O comando *scp* permite a cópia de arquivo entre cliente e servidor e também entre dois servidores remotos.
- ▶ A vantagem é que toda a comunicação é criptografada e protegida por senha.

Acessando um Servidor com SSH

Se você simplesmente fizer isso:

```
$ ssh 200.19.176.156
```

ou

```
$ ssh scripts.joao.marcelo.nom.br
```

- ▶ O cliente *ssh* vai tentar logar no servidor remoto (200.19.176.156 ou scripts.joao.marcelo.nom.br) usando o usuário que você está logado no sistema local.
- ▶ **Exemplo:** se seu usuário local for *alunoufc*, o cliente irá tentar logar como *alunoufc* na máquina remota.

Caso o usuário local não exista no servidor e deseje utilizar outro:

```
$ ssh joaomarclo@scripts.joao.marcelo.nom.br
```

Acessando um Servidor com SSH

Quando você acessa um servidor pela primeira vez:

```
The authenticity of host '18.213.179.235' can't be established.  
ECDSA key fingerprint is SHA256:LKdo00AkJT7haicaM.  
Are you sure you want to continue connecting (yes/no)?
```

- ▶ É apenas uma confirmação, digitando **yes** ela não aparece nas próximas conexões.
- ▶ Partimos para a parte de autenticação, que irá requisitar uma senha.
- ▶ Se a senha informada for correta, **sucesso**, você está logado no servidor.

Autenticação no Servidor SSH

Utilizar senhas tem seus problemas:

- ▶ Uma senha complexa é fácil de esquecer.
- ▶ O usuários acabam escolhendo senhas fáceis, que podem ser descobertas e causar problemas de segurança.
- ▶ Uma solução melhor e amplamente utilizada nas nuvens é o uso de um arquivo de identificação chamado **chave**.
- ▶ Cada arquivo de chave está associado com um usuário no servidor.
- ▶ O usuário deve ter cuidado para proteger sua chave, qualquer que tiver acesso à ela poderia acessar o servidor.

A solução mais segura é exigir que o usuário forneça uma **senha e a chave** toda vez que for logar. Mas deixamos para aprender isso mais adiante no curso.

Utilizando a Chave no SSH

```
$ ssh -i joamarcelo.pem joamarcelo@scripts.joao.marcelo.nom.br
```

- ▶ A chave é o arquivo *joamarcelo.pem*.
- ▶ O usuário remoto é *joamarcelo*. O nome do arquivo da chave e do usuário não precisam ser os mesmos.
- ▶ O servidor remoto é *scripts.joao.marcelo.nom.br*.
- ▶ O usuário *joamarcelo* deve existir no servidor remoto e estar configurado para permitir o acesso via chave.

No Slack do curso já enviei a chave privada de cada um. Podem tentar fazer o *login* agora.

Lembrete

A chave tem que ter permissões de acesso apenas para o usuário local. Portanto, confirme:

```
$ chmod 0600 joamarcelo.pem
```

Copiando Arquivos via SCP - Formato Geral

Copiando arquivos para o servidor

```
scp -i <chave> <arquivolocal> <usuario>@<servidor>:<diretorioemoto>
```

Copiando arquivos do servidor

```
scp -i <chave> <usuario>@<servidor>:<arquivoremoto> <arquivolocal>
```

Para copiar diretórios, podemos usar a opção *-r*.

Copiando Arquivos via SCP - Exemplos

Observação: os comandos abaixo podem ser digitados na mesma linha. Coloquei em várias linhas para caber no *slide*. A `\`(barra) pode ser usada para dividir um comando em várias linhas.

Copiando arquivos para o servidor

```
scp -i joaomarcelo.pem teste.txt \  
joaomarcelo@scripts.joao.marcelo.nom.br:/home/joaomarcelo/
```

Copiando arquivos do servidor

```
scp -i joaomarcelo.pem \  
joaomarcelo@scripts.joao.marcelo.nom.br:/home/joaomarcelo/teste.txt \  
teste.txt
```

Regras Gerais para Atividades

- ▶ **SEMPRE** leia com atenção todas as instruções.
- ▶ Obedeça observações sobre nome de pastas e arquivos no exercício.
- ▶ Se o exercício diz para criar uma pasta *atividade01*, crie a pasta com o mesmo nome. Não use *Atividade01*, *atividade1*, *aTivIDADE01*, etc.
- ▶ Na dúvida, pergunte ao professor.
- ▶ Muitas vezes usarei *scripts* para correção. Se os nomes estiverem diferentes, o *script* pode considerar a questão errada.

Atividade 01 - Atividade de como usar o SSH/SCP

1. Faça *login* no servidor `scripts.joao.marcelo.nom.br`.
2. Crie a pasta *atividades*.
3. Dentro da pasta *atividades*, crie o diretório *atividade01*.
4. Coloque o *script usuarios.sh* no diretório.
5. Execute o *script* no servidor.

Para colocar o *script* no servidor, você pode editá-lo localmente e copiá-lo em seguida ou já editá-lo no próprio servidor, usando o *nano* ou o *vim*.