Introdução

João Marcelo Uchôa de Alencar

Universidade Federal do Ceará - Quixadá

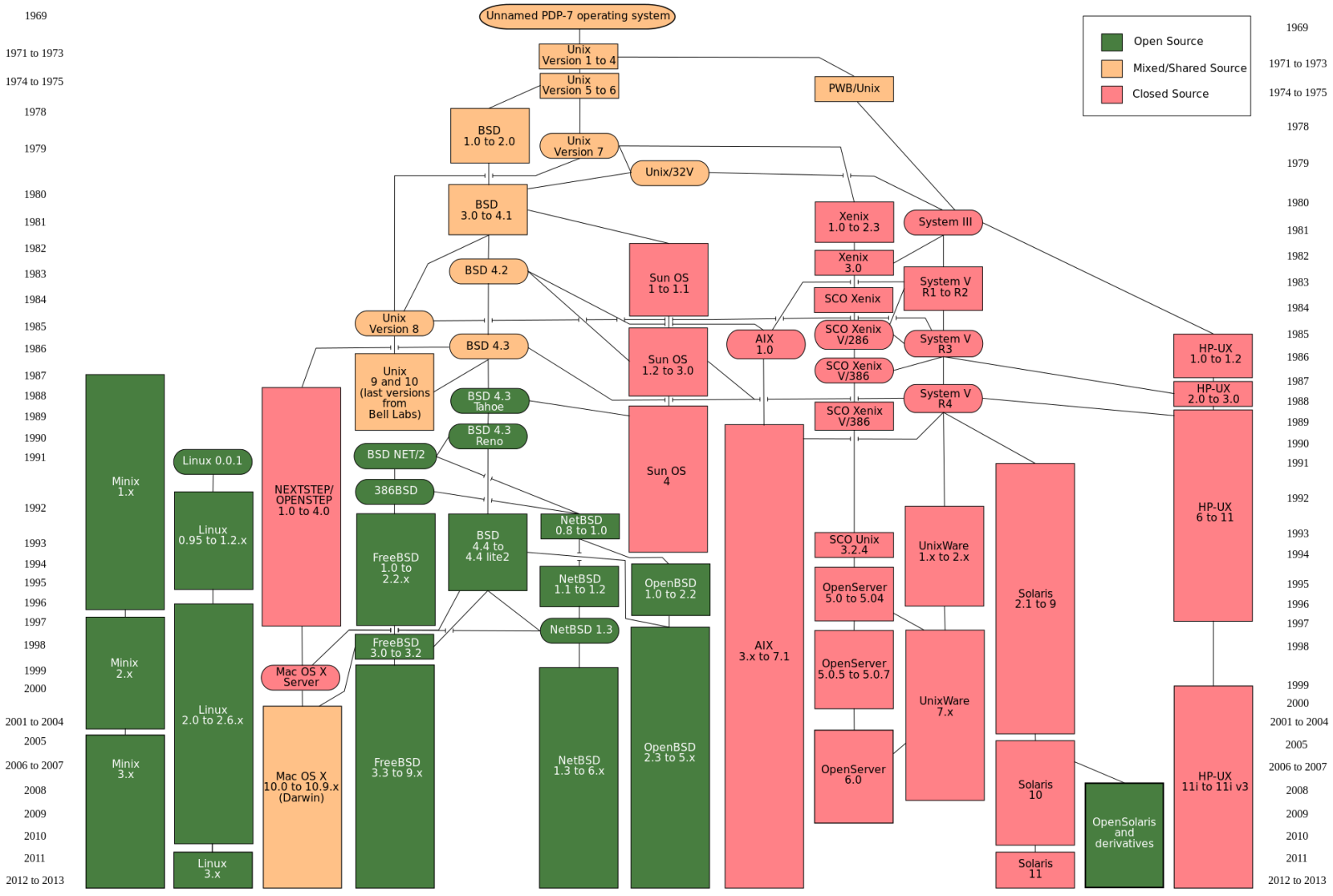
10 de agosto de 2023

Apresentação

- Professor João Marcelo Uchôa de Alencar.
- E-mail para contato: joao.marcelo@ufc.br.
- Grupo no Telegram.
- Avaliações:
  - 3 notas:
    - 3 grupos de atividades.
    - Não pode entregar vazia.
    - Atividades INDIVIDUAIS. Colar é proibido.
  - Fazer a média dessas 3 notas.
- Disciplina presencial, frequência dada por comparecimento à aula.

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.


Open Source
Mixed/Shared Source
Closed Source
1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2007
2008
2009
2010
2011
2012 to 2013
Unnamed PDP-7 operating system
Unix Version 1 to 4
Unix Version 5 to 6
PWB/Unix
BSD 1.0 to 2.0
Unix Version 7
Unix/32V
BSD 3.0 to 4.1
Xenix 1.0 to 2.3
System III
Xenix 3.0
BSD 4.2
Sun OS 1 to 1.1
SCO Xenix
System V R1 to R2
Unix Version 8
SCO Xenix V/286
AIX 1.0
System V R3
BSD 4.3
Sun OS 1.2 to 3.0
SCO Xenix V/386
HP-UX 1.0 to 1.2
Unix 9 and 10 (last versions from Bell Labs)
BSD 4.3 Tahoe
System V R4
HP-UX 2.0 to 3.0
BSD 4.3 Reno
BSD NET/2
Sun OS 4
HP-UX 6 to 11
386BSD
Minix 1.x
Linux 0.0.1
NEXTSTEP/ OPENSTEP 1.0 to 4.0
Linux 0.95 to 1.2.x
FreeBSD 1.0 to 2.2.x
BSD 4.4 to 4.4 lite2
NetBSD 0.8 to 1.0
AIX 3.x to 7.1
SCO Unix 3.2.4
UnixWare 1.x to 2.x
Solaris 2.1 to 9
NetBSD 1.1 to 1.2
OpenBSD 1.0 to 2.2
OpenServer 5.0 to 5.04
NetBSD 1.3
Minix 2.x
FreeBSD 3.0 to 3.2
OpenServer 5.0.5 to 5.0.7
UnixWare 7.x
Mac OS X Server
Linux 2.0 to 2.6.x
Minix 3.x
Mac OS X 10.0 to 10.9.x (Darwin)
FreeBSD 3.3 to 9.x
NetBSD 1.3 to 6.x
OpenBSD 2.3 to 5.x
OpenServer 6.0
Solaris 10
OpenSolaris and derivatives
HP-UX 11i to 11i v3
Linux 3.x
Solaris 11

Princípios Para Ferramentas de *Software*

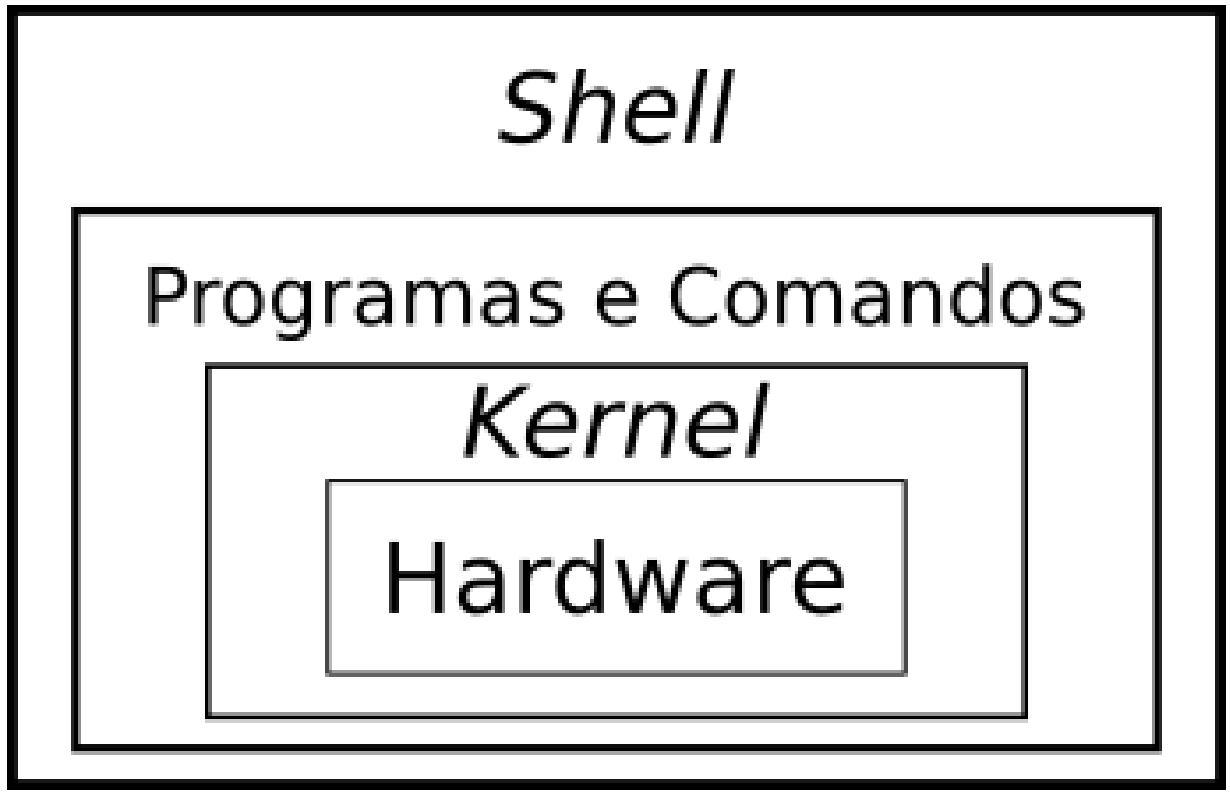
Os ambientes UNIX obedecem os seguintes princípios no desenvolvimento de suas ferramentas:

- 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**.
- Tente **generalizar** suas ferramentas.

Linguagens de *Scripting* versus Linguagens Compiladas

- Programas escritos em linguagens compiladas são traduzidos 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*?

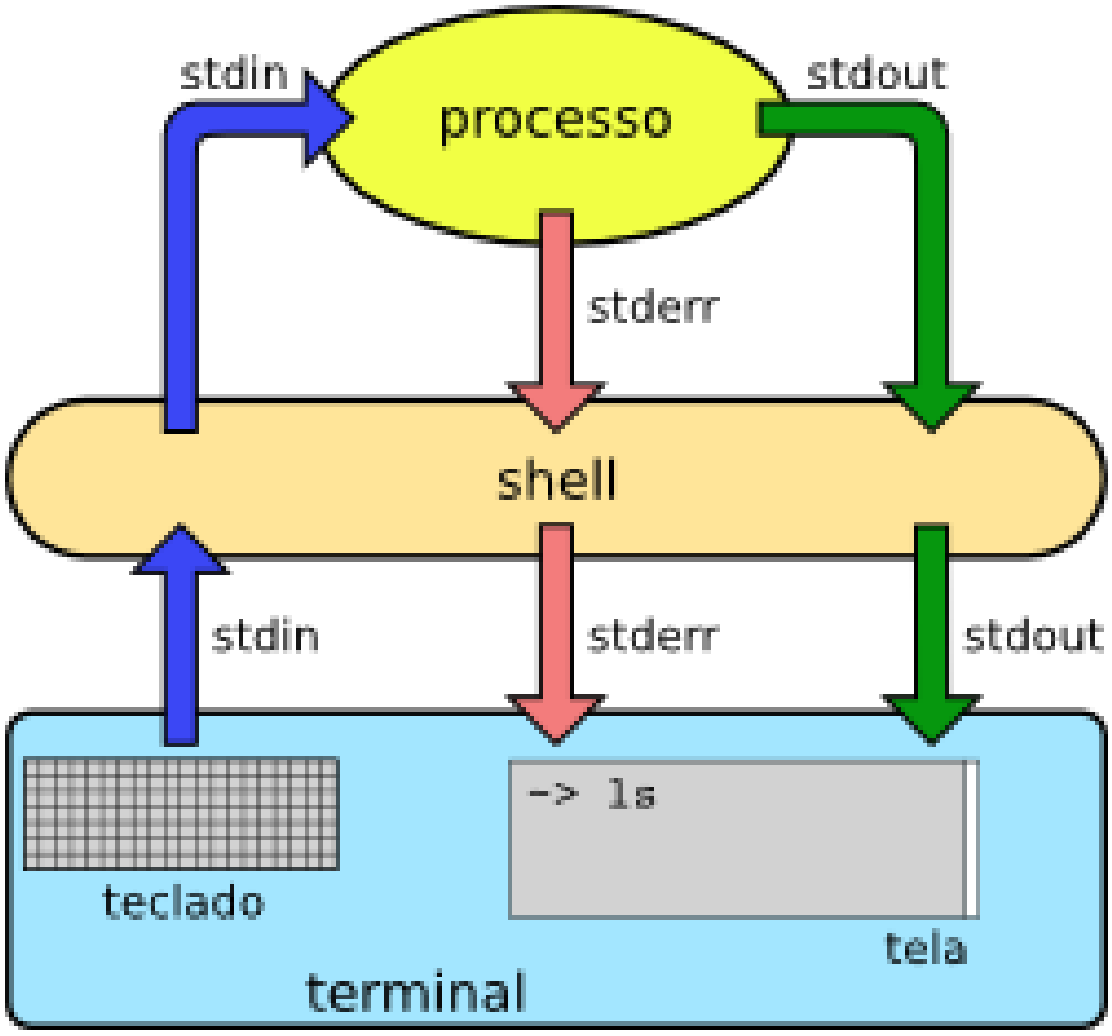
Ao nível do *Shell*:

```
$ 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 arquivos de texto chamados *scripts*, permitindo a automização 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).
- Z Shell (zsh).

Por que usar *Shell*?

- É mais fácil lidar com arquivos e diretórios em *shell*, assim 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.sh
```

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 programacaoscripts.joao.marcelo.nom.br
```

- O cliente *ssh* vai tentar logar no servidor remoto (200.19.176.156 ou programacaoscripts.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 joaomarcelo@programacaoscripts.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 poderiá 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 joaomarcelo.pem joaomarcelo@programacaoscripts.joao.marcelo.nom.br
```

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

Em nuvens computacionais (AWS, Azure, Google, etc) o *login* é feito por arquivo de chaves.

Lembrete

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

```
$ chmod 0600 joaomarcelo.pem
```

Copiando Arquivos via SCP - Formato Geral

Copiando arquivos para o servidor

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

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 abaixos 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@programacaoscripts.joao.marcelo.nom.br:/home/joaomarcelo/
```

Copiando arquivos do servidor

```
scp -i joaomarcelo.pem \
joaomarcelo@programacaoscripts.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.