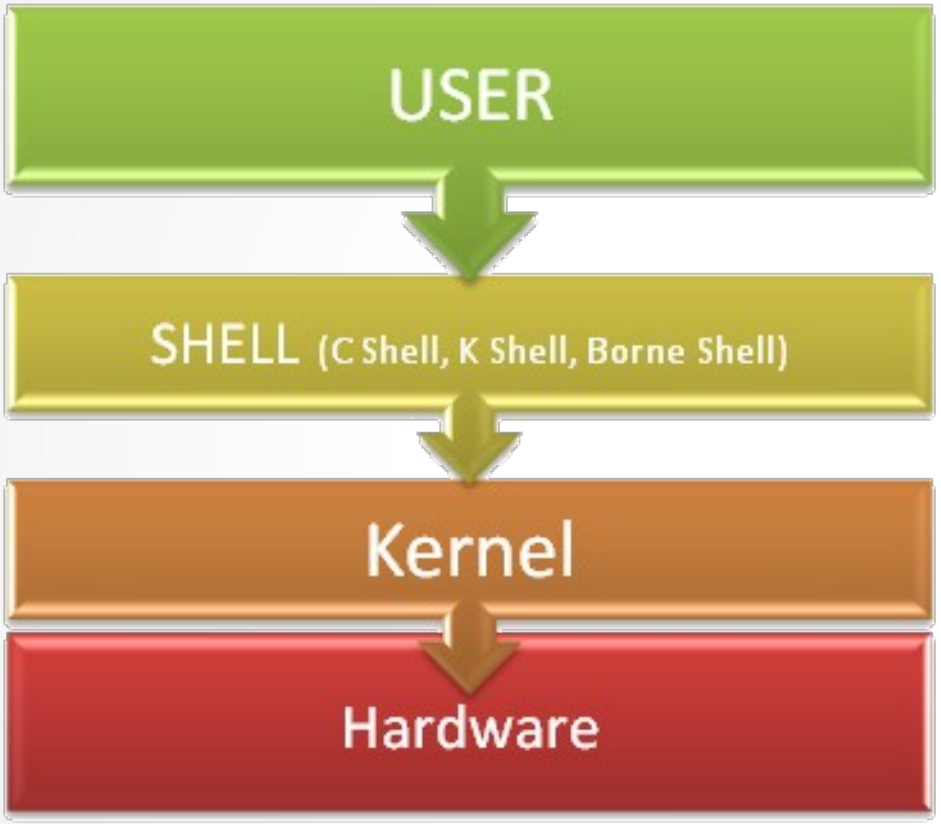
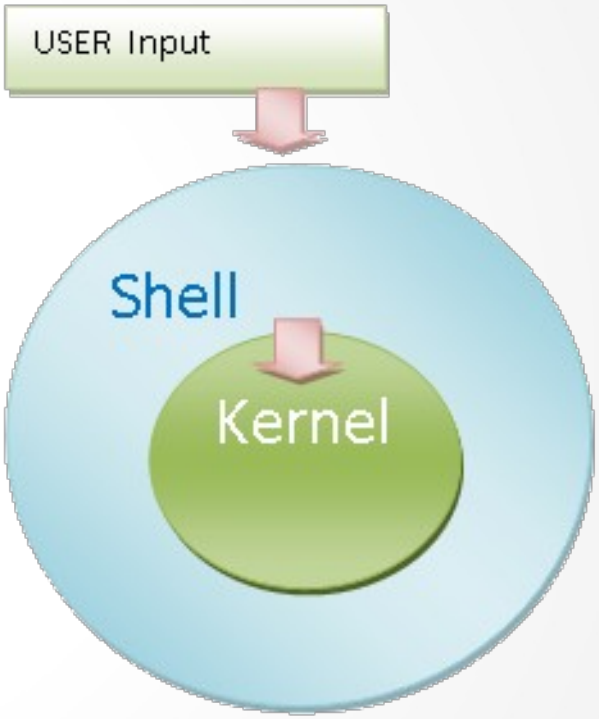
Introdução em Shell Script

Guilherme Magalhães Gall
gmgall@gmail.com

O que é o shell?





Alguns shells disponíveis no Linux

- bash
- ksh
- csh
- zsh
- sh

Como o shell interpreta a entrada do usuário

1) Análise da linha de comando
2) Resolução de redirecionamentos
3) Substituição de variáveis
4) Substituição de comandos
5) Substituição de meta-caracteres
6) Entrega para o kernel

Como o shell interpreta a entrada do usuário

1) Análise da linha de comando
2) Resolução de redirecionamentos
3) Substituição de variáveis
4) Substituição de comandos
5) Substituição de meta-caracteres
6) Entrega para o kernel

1. Análise da linha de comando

- Nessa etapa o shell identifica os caracteres especiais e verifica se a linha é um comando ou definição de variável.

Comando

- Divide a linha em pedaços separados por espaços em branco; o 1º é o comando, os demais, parâmetros.
- Verifica se o programa existe e as permissões.
- Resolve/substitui redirecionamentos e variáveis.
- Atenção: espaços em branco repetidos são removidos.

Comando

```
$ ls /
bin    etc             lib         mnt   run      sys  vmlinuz
boot   home            lib64       opt   sbin     tmp  vmlinuz.old
cdrom  initrd.img      lost+found  proc  selinux  usr
dev    initrd.img.old  media       root  srv      var

$     ls       / # remove espaços repetidos
bin    etc             lib         mnt   run      sys  vmlinuz
boot   home            lib64       opt   sbin     tmp  vmlinuz.old
cdrom  initrd.img      lost+found  proc  selinux  usr
dev    initrd.img.old  media       root  srv      var

$ ls /root
ls: não foi possível abrir o diretório /root: Permissão negada
```

Definição de variáveis

- Se o shell encontra 2 campos separados por igual (=), considera a linha uma definição de variável.
- Não deve haver espaços em branco entre eles ou a linha será tratada como comando.

Definição de variáveis

```
$ # define var com valor 10
$ var=10

$ # tenta executar comando var com 2 parâmetros: = e 10
$ var = 10
var: command not found
```

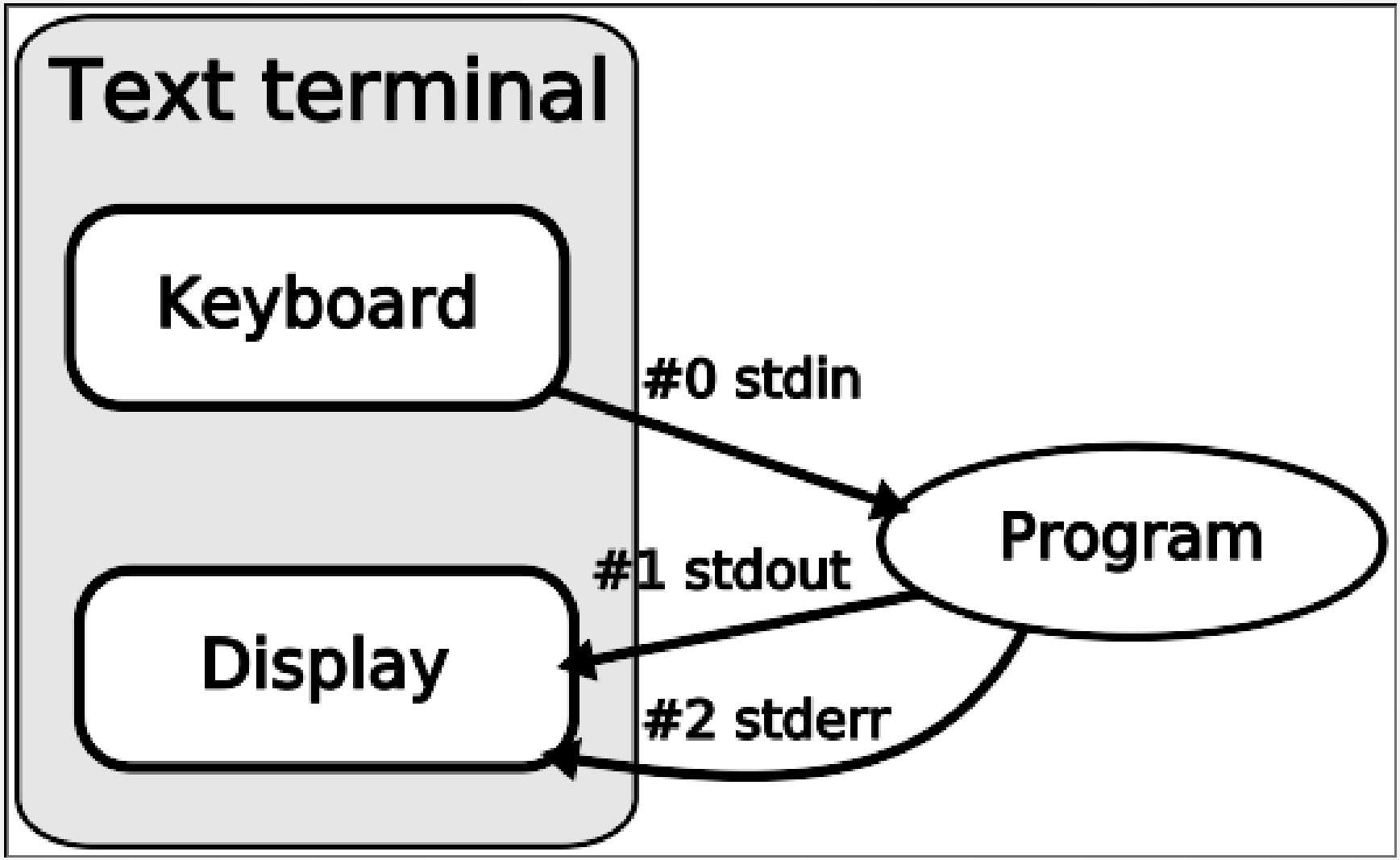
Como o shell interpreta a entrada do usuário

1) Análise da linha de comando
2) Resolução de redirecionamentos
3) Substituição de variáveis
4) Substituição de comandos
5) Substituição de meta-caracteres
6) Entrega para o kernel

2. Resolução de redirecionamentos

- Os comandos tem uma entrada (`stdin`), uma saída (`stdout`) e podem gerar erros (`stderr`).
- Dispositivos padrões:
  - `stdin`: teclado do terminal
  - `stdout`: vídeo do terminal
  - `stderr`: vídeo do terminal
- Isso pode ser alterado com redirecionamentos.

2. Resolução de redirecionamentos



Redirecionamentos de saída

- A saída (`stdout`) pode ser redirecionada para um arquivo.
- `comando > arquivo`
  - limpa `arquivo` e redireciona a saída de `comando` para `arquivo`
- `comando >> arquivo`
  - adiciona saída de `comando` a `arquivo`

Redirecionamentos de saída

```
$ echo A data de hoje é:
A data de hoje é:

$ date
Sáb Jan 19 15:43:03 BRST 2013

$ echo A data de hoje é: > hoje.txt
$ date >> hoje.txt
$ cat hoje.txt
A data de hoje é:
Sáb Jan 19 15:44:25 BRST 2013
```

Redirecionamentos de saída

```
$ rm hoje.txt

$ echo A data de hoje é: > hoje.txt

$ date > hoje.txt

$ cat hoje.txt
Sáb Jan 19 15:46:52 BRST 2013
```

Cuidado com os redirecionamentos! (1)

```
$ cat chamada.txt
Arthur
Zaphod
Trillian
Ford
Marvin

$ sort chamada.txt
Arthur
Ford
Marvin
Trillian
Zaphod
```

Cuidado com os redirecionamentos! (2)

```
$ sort chamada.txt > chamada.txt

$ cat chamada.txt # arquivo vazio!
```

```
$ sort chamada.txt >> chamada.txt
$ cat chamada.txt
Arthur
Zaphod
Trillian
Ford
Marvin
Arthur
Ford
Marvin
Trillian
Zaphod
```

Cuidado com os redirecionamentos! (3)

Por que isso aconteceu?

Lembre-se que os redirecionamentos (2) são resolvidos antes da execução dos comandos (6).

Redirecionamentos de erros

- Os erros (`stderr`) podem ser redirecionados para um arquivo.
- `comando 2> arquivo`
  - limpa `arquivo` e redireciona a saída de erros de `comando` para `arquivo`
- `comando 2>> arquivo`
  - adiciona saída de erros de `comando` a `arquivo`

Redirecionamentos de erros

```
$ cd ~/vpn_lncc/usr/

$ grep After -r .
./usr.crt: Not After : May 20 19:54:15 2013 GMT
grep: ./clienteVPN.log: Permissão negada

$ echo Erros: > erros.txt
$ grep After -r . 2>> erros.txt
./usr.crt: Not After : May 20 19:54:15 2013 GMT

$ cat erros.txt
Erros:
grep: ./clienteVPN.log: Permissão negada
```

Redirecionamentos de erros

```
$ echo Erros: > erros.txt
$ grep After -r . 2> erros.txt
./usr.crt: Not After : May 20 19:54:15 2013 GMT

$ cat erros.txt
grep: ./clienteVPN.log: Permissão negada
```

Redirecionamentos de saída + erros

- É possível redirecionar a saída (`stdout`) e os erros (`stderr`) para um arquivo.
- `comando &> arquivo`
  - limpa `arquivo` e redireciona a saída + erros de `comando` para `arquivo`
- `comando &>> arquivo`
  - adiciona saída + erros de `comando` a `arquivo`

Redirecionamentos de saída + erros

```
$ cd ~/vpn_lncc/usr/
$ grep After -r .
./usr.crt: Not After : May 20 19:54:15 2013 GMT
grep: ./clienteVPN.log: Permissão negada

$ grep After -r . &> /tmp/registro

$ cat /tmp/registro
./usr.crt: Not After : May 20 19:54:15 2013 GMT
grep: ./clienteVPN.log: Permissão negada
```

/dev/null

- Se a saída padrão (`stdout`) ou a saída de erros (`stderr`) não interessarem, é possível omiti-la com um redirecionamento para `/dev/null`
- É um arquivo especial que não retém nada do que é escrito nele.
- “Buraco negro“ do Unix.

/dev/null

```
$ cd ~/vpn_lncc/usr/
$ grep After -r .
./usr.crt: Not After : May 20 19:54:15 2013 GMT
grep: ./clienteVPN.log: Permissão negada

$ grep After -r . 2> /dev/null
./usr.crt: Not After : May 20 19:54:15 2013 GMT
```

Redirecionamentos de entrada

- A entrada padrão (`stdin`) pode vir de um arquivo.
- `comando < arquivo`
  - `comando` receberá a entrada de `arquivo` ao invés de receber do teclado

Redirecionamentos de entrada

```
$ cat chamada.txt
Arthur
Zaphod
Trillian
Ford
Marvin

$ tr a-z A-Z < chamada.txt
ARTHUR
ZAPHOD
TRILLIAN
FORD
MARVIN
```

Redirecionamentos de entrada

- A entrada padrão (`stdin`) pode ser lida até que o shell encontre um `<<MARCADOR`
-
```
comando <<FIM
texto
multilinha
aqui
FIM
```
- O shell considerará até `FIM` a entrada padrão de `comando`
- here-documents

Redirecionamentos de entrada

```
$ sed 's|http://||' <<FIM
> http://www.gnu.org/software/bash/
> http://www.kornshell.com/
> http://www.zsh.org/
> FIM
www.gnu.org/software/bash/
www.kornshell.com/
www.zsh.org/
```

Combinando redirecionadores

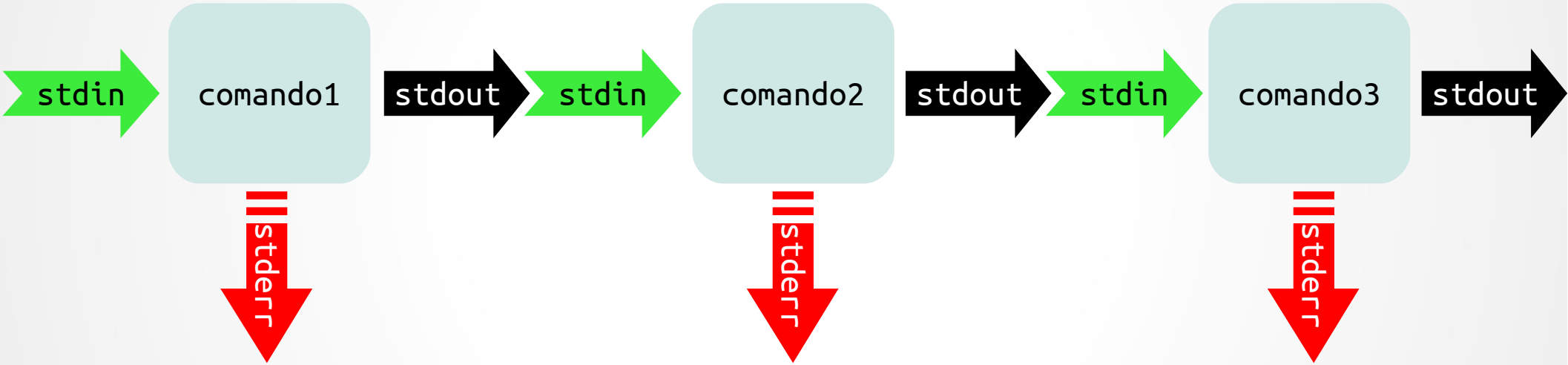
```
$ sed 's|http://||' <<FIM > lista_sites
http://www.gnu.org/software/bash/
http://www.kornshell.com/
http://www.zsh.org/
FIM

$ cat lista_sites
www.gnu.org/software/bash/
www.kornshell.com/
www.zsh.org/
```

pipe (|)

- Redireciona a saída padrão (`stdout`) de um comando para a entrada padrão (`stdin`) de outro.
- Permite encadear comandos com especialidades diversas para resolver um problema específico.
- Analogia com uma linha de montagem.
- Sintaxe: `comando1 | comando2 | comando3`

pipe (|)


stdin
comando1
stdout
stdin
comando2
stdout
stdin
comando3
stdout
stderr
stderr
stderr

pipe (|)

- Exemplo: obter uma lista dos shells dos usuários do sistema.
- O arquivo `/etc/passwd` lista os logins do sistema, um por linha.
- Exemplo de uma linha:

  `root:x:0:0:root:/root:/bin/bash`
- O 7º campo contém o shell do usuário.

pipe (|)

```
$ cut -d: -f 7 /etc/passwd
/bin/bash
/bin/sh
/bin/sh
/bin/sh
/bin/sync
/bin/sh
/bin/sh
/bin/sh
/bin/sh
/bin/sh
/bin/sh
/bin/sh
/bin/sh
/bin/sh
...
```

pipe (|)

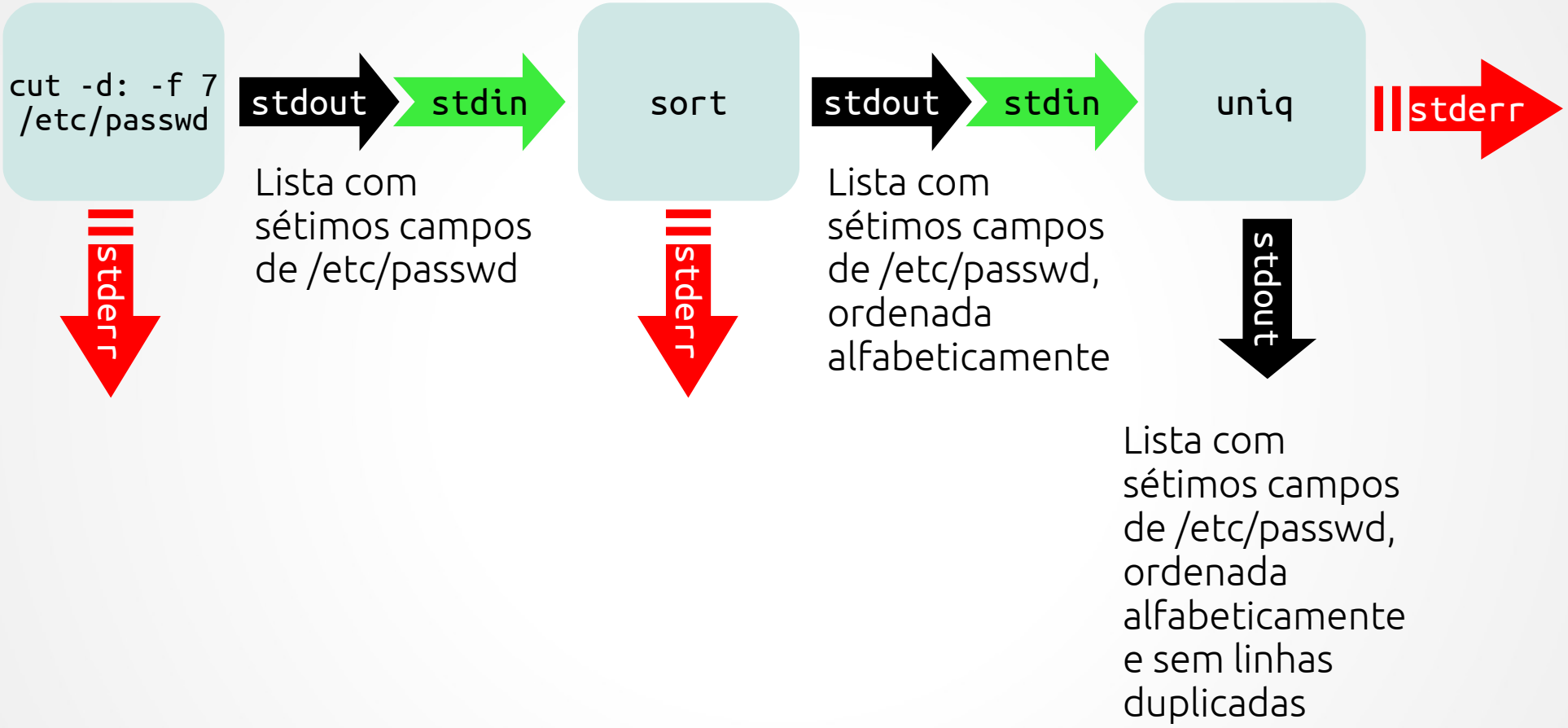
```
$ cut -d: -f 7 /etc/passwd | uniq
/bin/bash
/bin/sh
/bin/sync
/bin/sh
/bin/false
/bin/sh
/bin/false
/bin/bash
/bin/false
/usr/sbin/nologin
/bin/false
```

pipe (|)

```
$ cut -d: -f 7 /etc/passwd | sort
/bin/bash
/bin/bash
/bin/false
/bin/false
/bin/false
/bin/false
…

$ cut -d: -f 7 /etc/passwd | sort | uniq
/bin/bash
/bin/false
/bin/sh
/bin/sync
/usr/sbin/nologin
```

pipe (|)


cut -d: -f 7 /etc/passwd
stdout
stdin
sort
stdout
stdin
uniq
stderr
stderr
stderr
stdout
Lista com sétimos campos de /etc/passwd
Lista com sétimos campos de /etc/passwd, ordenada alfabeticamente
Lista com sétimos campos de /etc/passwd, ordenada alfabeticamente e sem linhas duplicadas

Como o shell interpreta a entrada do usuário

1)Análise da linha de comando
2)Resolução de redirecionamentos
3)Substituição de variáveis
4)Substituição de comandos
5)Substituição de meta-caracteres
6)Entrega para o kernel

3. Substituição de variáveis

- Nessa etapa o shell substitui os `$variavel` da linha pelo conteúdo da variável.

```
$ var=100
$ echo O valor de var é $var
O valor de var é 100
```

- Bom momento para apresentar os caracteres de remoção de significado, que protegem uma string da interpretação do shell.

Contrabarra \

- Protege apenas o caractere que a sucede.

```
$ var=100
$ echo Protegendo \$var de interpretação
Protegendo $var de interpretação
```

Aspas simples '

- Protege a sequência de caracteres entre elas.

```
$ echo 'Protegendo $var de interpretação'
Protegendo $var de interpretação

$ echo Protegendo '$var' de interpretação
Protegendo $var de interpretação
```

Aspas duplas "

- Protege a sequência de caracteres entre elas.
- Porém, se a sequência contiver um cifrão ($), crase (`) ou contrabarra (\), esses caracteres serão interpretados.

```
$ var=100
$ echo "Use \$var para mostrar o valor $var"
Use $var para mostrar o valor 100

$ # a contrabarra foi interpretada pelo shell
```

Como o shell interpreta a entrada do usuário

1) Análise da linha de comando
2) Resolução de redirecionamentos
3) Substituição de variáveis
4) Substituição de comandos
5) Substituição de meta-caracteres
6) Entrega para o kernel

4. Substituição de comandos

- Permite que a saída padrão de um comando ou pipeline substitua seu nome.
- `$(comando)` ou `` `comando` ``
- A 1ª forma é preferível.
- http://mywiki.wooledge.org/BashFAQ/082

4. Substituição de comandos

```
$ lista=$(ls -l)
$ echo O diretório $(pwd) contém: $lista
O diretório /home/guilherme contém: total 60 drwxr-xr-x 4 guilherme guilh
$ # Saiu tudo numa única linha!

$ echo -e "O diretório $(pwd) contém:\n$lista"
Listando o diretório /home/guilherme:
total 60
drwxr-xr-x  4 guilherme guilherme 4096 Jan 19 22:02 Área de Trabalho
-rw-rw-r--  1 guilherme guilherme   35 Jan 19 16:47 chamada.txt
drwxr-xr-x  2 guilherme guilherme 4096 Abr 27  2012 Documentos
drwxr-xr-x 34 guilherme guilherme 4096 Dez 28 16:04 Downloads
drwx------  8 guilherme guilherme 4096 Jan 20 14:51 Dropbox
drwxr-xr-x  2 guilherme guilherme 4096 Dez  2 04:45 Imagens
-rw-rw-r--  1 guilherme guilherme   59 Jan 19 17:30 lista_sites
drwxr-xr-x  2 guilherme guilherme 4096 Abr 27  2012 Modelos
drwxr-xr-x  5 guilherme guilherme 4096 Jan  9 21:59 Música
drwxr-xr-x  2 guilherme guilherme 4096 Out  1 13:14 Público
drwxrwxr-x  4 guilherme guilherme 4096 Jun  3  2012 repos
drwxr-xr-x  2 guilherme guilherme 4096 Abr 27  2012 Vídeos
drwxrwxr-x  3 guilherme guilherme 4096 Mai 18  2012 vpn_lncc
```

Dica: (quase) sempre use aspas

- Na declaração de variáveis, argumentos para comandos e ao redor de variáveis sendo referenciadas. Isso evitará muitos problemas.
- Só não use quando desejar que o shell elimine os espaços em branco consecutivos. Essas situações são raras.

Como o shell interpreta a entrada do usuário

1) Análise da linha de comando
2) Resolução de redirecionamentos
3) Substituição de variáveis
4) Substituição de comandos
5) Substituição de meta-caracteres
6) Entrega para o kernel

5. Substituição de meta-caracteres

- Nessa etapa, se o shell encontra qualquer meta-caractere na linha, ele o substitui por seus possíveis valores.
- Também chamados de coringas

Coringa	Casa com...
*	Qualquer coisa
?	Um caractere qualquer
[Aa]	Qualquer um dos caracteres listados
[^Aa]	Qualquer caractere, menos os listados
.	Diretório atual
..	Diretório superior
~	Home do usuário

5. Substituição de meta-caracteres

```
$ ls
arquivo1.txt  Arquivo1.txt  Arquivo2  arquivo2.txt

$ ls *txt
arquivo1.txt  Arquivo1.txt  arquivo2.txt

$ ls arquivo?.txt
arquivo1.txt  arquivo2.txt

$ ls [Aa]rquivo?.txt
arquivo1.txt  Arquivo1.txt  arquivo2.txt

$ ls Arquivo[^1]
Arquivo2
```

5. Substituição de meta-caracteres

```
$ pwd
/tmp/exemplo
$ cd ..
$ pwd
/tmp

$ mv exemplo/[Aa]rquivo2* .
$ ls exemplo/
arquivo1.txt  Arquivo1.txt

$ ls ?rquivo*
Arquivo2  arquivo2.txt
$ # Perceba que * casa qualquer coisa,
$ # inclusive nada.
```

Como o shell interpreta a entrada do usuário

1) Análise da linha de comando
2) Resolução de redirecionamentos
3) Substituição de variáveis
4) Substituição de comandos
5) Substituição de meta-caracteres
6) Entrega para o kernel

6. Entrega para o kernel

- Com as etapas anteriores concluídas, o shell monta a linha de comando com todas as substituições feitas e chama o kernel para executá-la em um novo shell filho.
- O shell pai fica aguardando inativo enquanto o programa é executado.
- Quando o shell filho termina, o controle volta para o pai, que exibe o prompt novamente, indicando que está pronto para novos comandos.