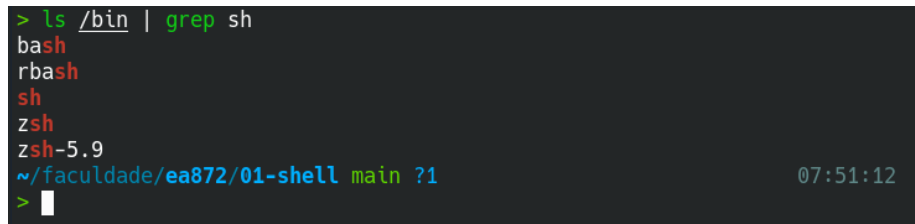


Relatório 01

Vinícius de Oliveira Peixoto Rodrigues (245294)

Agosto de 2022

Questão 1



```
> ls /bin | grep sh
bash
rbash
sh
zsh
zsh-5.9
~/faculdade/ea872/01-shell main ?1
>
```

Figura 1: Lista de *shells* instaladas no sistema

As *shells* no meu sistema são instaladas pelo *package manager* em `/bin`, de modo que um `grep` é o suficiente para encontrar as *shells* instaladas. A minha distribuição vem por padrão com `bash` e `sh`, e o `zsh` foi instalado por mim.

Questão 2

```
(desktop:~/faculdade/ea872/01-shell) nuke% ps -a | grep $$
29694 pts/2    00:00:00 csh
(desktop:~/faculdade/ea872/01-shell) nuke% ls
hello.sh latex
(desktop:~/faculdade/ea872/01-shell) nuke% echo $PATH
/usr/local/texlive/2022/bin/x86_64-linux/:/home/nuke/.local/bin:/home/nuke/.cargo/bin:/home/nuke/.cargo/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/opt/bin:/usr/lib/llvm/14/bin:/opt/cuda/bin:/usr/lib64/opencascade/bin
(desktop:~/faculdade/ea872/01-shell) nuke% setenv PATH $PWD\: $PATH
(desktop:~/faculdade/ea872/01-shell) nuke% echo $PATH
/home/nuke/faculdade/ea872/01-shell:/usr/local/texlive/2022/bin/x86_64-linux/:/home/nuke/.local/bin:/home/nuke/.cargo/bin:/home/nuke/.cargo/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/opt/bin:/usr/lib/llvm/14/bin:/opt/cuda/bin:/usr/lib64/opencascade/bin
(desktop:~/faculdade/ea872/01-shell) nuke% hello.sh
Hello world!
(desktop:~/faculdade/ea872/01-shell) nuke% █
```

Figura 2: Adicionando um novo caminho ao \$PATH do csh

Para modificar uma *environment variable* no `csh`, utiliza-se o comando

```
setenv <var> <value>
```

A Figura 2 ilustra esse processo adicionando um script `hello.sh` no diretório local, concatenando em seguida o `$PWD` ao `$PATH` da *shell*.

Questão 3

A tabela abaixo contém os valores, assim como uma explicação breve sobre cada um deles:

Variável	Valor	Explicação
\$0	prog	nome do arquivo
\$2	le-27	segundo argumento
\$4	unicamp	quarto argumento
\$8	brasil	oitavo argumento
\$\$	<número>	pid da <i>shell</i>
\$#	11	numero de argumentos
\$*	prog le-27 feec unicamp campinas são paulo...	concatenação dos argumentos
@	prog le-27 feec unicamp campinas são paulo...	lista dos argumentos

Questão 5

```
#!/bin/sh

echo menu # printa "menu"
stop=0    # inicializa flag
while test $stop -eq 0 # enquanto flag != 0
do
    echo          # imprime newline

    # here-document (usa o texto desse script como stdin pro `cat`)
    # ou seja, printa essa multi-line string
    cat <<FIMDOMENU
1 : imprime a data
2,3 : imprime o diretorio corrente
4 : fim
FIMDOMENU
    echo          # imprime newline
    echo 'opcao? ' # imprime "opcao? "
    read op       # le do stdin
    echo          # imprime newline
    case $op in
        1) date;; # faz match da entrada com 1, 2, 3, 4
        # imprime a data
        2|3) pwd;; # imprime o diretorio atual
        4) stop=1;; # seta a flag de parada
        *) echo opcao invalida! ;; # erro caso nao haja match
    esac
done
```

Figura 3: Item (a)

```

case $# in
0) set `date`; m=$2; y=$6; # faz match no numero de argumentos
# se sem argumentos: usa a data atual, `date` = ${date}
# da match no valor em EN de m e troca pra PT
case $m in
Feb) m=Fev;;
Apr) m=Abr;;
May) m=Mai;;
Aug) m=Ago;;
Sep) m=Set;;
Oct) m=Out;;
Dec) m=Dez;;
esac;;
1) m=$1; set `date`; y=$6;; # se 1 argumento: guarda o mes, pega o ano com `date`
*) m=$1; y=$2 ;; # se 2 argumentos: guarda mes e ano
esac
case $m in
jan*|Jan*) m=1;;
fev*|Fev*) m=2;;
mar*|Mar*) m=3;;
abr*|Abr*) m=4;;
mai*|Mai*) m=5;;
jun*|Jun*) m=6;;
jul*|Jul*) m=7;;
ago*|Ago*) m=8;;
set*|Set*) m=9;;
out*|Out*) m=10;;
nov*|Nov*) m=11;;
dez*|Dez*) m=12;;
[1-9]|10|11|12) ;; # se ja e um numero, nao faz nada
*) y=$m; m="" ;; # se sem match, guarda o valor vazio pro mes
esac
/usr/bin/cal $m $y # printa a data no calendario (comando `cal`)

```

Figura 4: Item (b)

```

#!/bin/sh
for DIRPATH in `echo $PATH | sed 's:/ /g'` # troca o separador ":" do $PATH por um whitespace
do # de modo a conseguir iterar por eles como uma lista
if [ ! -d $DIRPATH ] # 1: se $DIRPATH nao (!) e um diretorio (-d)
then
if [ -f $DIRPATH ] # 2: se $DIRPATH eh um arquivo (-f)
then
echo "$DIRPATH nao e diretorio, e um arquivo" # 2: printa erro, e um arquivo
else # 2: caso contrario...
echo "$DIRPATH nao existe" # 2: printa erro, nao existe
fi # 2: fim do condicional
fi # 1: fim do condicional
done

```

Figura 5: Item (c)

```

#!/bin/sh

case $# in # checa se o numero de argumentos eh 2
    # caso contrario printa erro no stderr e sai com codigo de erro 2
    # (misuse of shell command)
    0|1|[3-9]) echo  Uso: classifica arquivo1 arquivo2  1>&2; exit 2 ;;
esac
total=0; perdid=0;
while read novalinha # le do stdin ate encontrar o EOF
do total=`expr $total + 1` # incrementa o contador de linhas lidas
    case "$novalinha" in          # da match na linha
        # se possui somente letras, escreve linha em arquivo1
        *[A-Za-z]*) echo "$novalinha" >> $1 ;;
        # se possui somente numeros, escreve linha em arquivo2
        *[0-9]*) echo "$novalinha" >> $2 ;;
        # se encontrar a sequencia "<>", interrompe loop
        '<>') break;;
        # caso contrario, nao escreve a linha e incrementa o contador de linhas perdidas
        *) perdid=`expr $perdid + 1`;;
    esac
done

# printa o total de linhas lidas (e escritas em arquivos) e o de linhas perdidas
# faz $total - 1 porque o "<>" incrementa o contador mas nao e aproveitado
echo "`expr $total - 1` linha(s) lida(s), $perdid linha(s) nao aproveitada(s)"

```

Figura 6: Item (d)

Questão 6

O programa registra *signal handlers* para os sinais SIGTERM e SIGINT, depois procede a entrar em *idle* e checar a cada 5 segundos se recebeu um *signal* do sistema, imprimindo uma mensagem quando detectar o recebimento de um.

```
#!/bin/sh

ARQUIVO=arq.$$
touch $ARQUIVO # cria uma arquivo arq.PID, com o PID do processo

# registra traps (um comando a ser executado mediante o recebimento de um
# determinado signal, mesmo principio que o sighandler_t do signal.h)

# trap para SIGTERM, deleta arq.PID e printa erro
trap "echo 'Algum processo enviou um TERM' 1>&2; rm -f $ARQUIVO; exit;" 15
# trap para SIGINT, deleta arq.PID e printa erro
trap "echo 'Algum processo enviou um INT' 1>&2; rm -f $ARQUIVO; exit;" 2

# loop infinito, checa a cada 5 segundos se recebeu algum sinal
while true
do
    # Espera 5 segundos
    sleep 5
done
```

Figura 7: Explicação detalhada do script `traps`

```
> ./traps &
[1] 519
> kill -2 519
~/faculdade/ea872/01-shell/files/arqlab1/atividades main !3 ?3 % 11:56:24
> Algum processo enviou um INT

[1] + 519 done ./traps
> ./traps &
[1] 545
> kill -15 545
~/faculdade/ea872/01-shell/files/arqlab1/atividades main !3 ?3 % 11:56:37
> Algum processo enviou um TERM

[1] + 545 done ./traps
~/faculdade/ea872/01-shell/files/arqlab1/atividades main !3 ?3 11:56:37
>
```

Figura 8: Execução do script `traps`

Questão 7

```
#!/bin/sh

# se os argumentos $1, $2, ... existirem (nao forem nulos),
# sao armazenados em param1, param2, ...
test -n "$1" && param1=$1
test -n "$2" && param2=$2
test -n "$3" && param3=$3
test -n "$4" && param4=$4

# se param1 nao for setado, usa "rs"
echo "1o resultado do teste:${param1-rs} com param1 = $param1"
# se param2 nao for setado, usa "pa" e faz param2="pa"
echo "2o resultado do teste:${param2=pa} com param2 = $param2"
# se param3 FOR setado, usa "to"
echo "3o resultado do teste:${param3+to} com param3 = $param3"
# se param4 nao for setado, imprime a mensagem e sai com status de erro
echo "4o resultado do teste:${param4?Quarto parâmetro não iniciado} com param4 = $param4"
```

Figura 9: Explicação detalhada do script `subspar`

```
> ./subspar sp rj mg es df pr mt ms
1o resultado do teste:sp com param1 = sp
2o resultado do teste:rj com param2 = rj
3o resultado do teste:to com param3 = mg
4o resultado do teste:es com param4 = es
~/faculdade/ea872/01-shell/files/arqlab1/atividades main !3 ?3
>
```

Figura 10: Execução do script `subspar`

A Figura 9 contém uma explicação passo a passo do script `subspar`. A Figura 10 contém a execução do script com as entradas dadas. Percebe-se que as entradas `param1`, `param2` e `param4` se mantiveram inalteradas (porque estava setadas), enquanto `param3` foi impresso como `to` porque `param3` estava setado.

Questão 8

```
#!/bin/sh

# uso:
# pro_lixo l: lista os arquivos na lixeira ($HOME/lixo)
# pro_lixo r: "esvazia" a lixeira (deleta tudo dentro de $HOME/lixo)
# pro_lixo [FILE]...: move uma lista de arquivos pra lixeira
#
# serve como uma alternativa ao `rm`, movendo arquivos para a lixeira em vez
# de deleta-los imediatamente

# testa se a lixeira ja existe, caso contrario cria o diretorio
test -d $HOME/lixo || mkdir $HOME/lixo
# caso nao tenha argumentos, aborta com status 1
test 0 eq "$#" && exit 1;

case $1 in
l) ls $HOME/lixo;; # l -> lista os arquivos da lixeira
r) case $# in
# se `pro_lixo r`, salva o dir atual, entra na lixeira, deleta tudo, volta pro dir original
1) aux=$PWD; cd $HOME/lixo; rm -rf *; cd $aux;;
*) echo pro_lixo: Uso incorreto;; # se argc != 1 (`pro_lixo r`), printa erro
esac;;
# itera pela lista e move os arquivos pra lixeira
*) for i in $*
do
if test -f $i # checa se o arquivo existe
then mv $i $HOME/lixo # se sim, move pra lixeira
else echo pro_lixo: Arquivo $i nao encontrado. # caso contrario printa erro
fi
done;;
esac
```

Figura 11: Explicação detalhada do script `pro_lixo`

Como detalhado na Figura 11, a utilidade prática do script é mover arquivos para a lixeira (uma pasta localizada em `$HOME/lixo`).

Questão 9

O script é basicamente uma implementação simplificada do comando **tree**, que imprime os arquivos dentro de uma pasta no formato de uma árvore. Abaixo seguem comentários detalhados do script (fiz uma pequena modificação para fazer ele funcionar sem estar no **\$PATH**, de modo a conseguir fazer rodar) e também uma imagem mostrando seu funcionamento:

```
#!/bin/sh

SCRIPT_PATH=$(realpath "$0")

if [ $# -eq 0 ] # se não há argumentos...
then
    set $PWD # seta $1 para $PWD
fi

for ARG in $* # itera pela arglist
do
    case $ARG in
        --prof*) # da match nos argumentos
            # extrai o valor de profundidade (numero), do argumento
            # separando em dois grupos (--prof, <numero>) com separador
            # =
            PROFUNDIDADE=$(echo $ARG | cut -f 2 -d '=')
            ;;
        *)
            # se $ARG é um diretório...
            if [ -d $ARG ]
            then
                # $CONT recebe o valor de $PROFUNDIDADE (ou 0, caso
                # não esteja setado)
                CONT=$((PROFUNDIDADE+0))
                # enquanto $CONT é maior que 0...
                while [ $CONT -gt 0 ]
                do
                    # printa whitespaces para "indentar" a árvore
                    # de diretórios
                    # 2 whitespaces = 1 nível de profundidade
                    echo -n "  "
                    CONT=$((CONT - 1))
                done
                # printa o nome do diretório
                echo -n "$ARG"
                # entra no diretório
                cd $ARG
                # chama a si mesmo recursivamente para cada arquivo
                # no diretório, com profundidade incrementada de 1
                for NAME in *
                do
                    ${SCRIPT_PATH} --prof=$(expr $PROFUNDIDADE + 1) $NAME
                done
            else
                if [ -f $ARG ]
                then
                    # mesma ideia, printa whitespaces para
                    # "indentar" a árvore, mas dessa vez não
                    # chama a si mesmo recursivamente (visto que
                    # é um arquivo e não uma pasta)
                    CONT=$((PROFUNDIDADE+0))
                    while [ $CONT -gt 0 ]
                    do
                        echo -n "  "
                        CONT=$((CONT - 1))
                    done
                    echo -n "$ARG"
                fi
            fi
        esac
    done
done

arqlab1/atividades/tree 2,0-1 Top arqlab1/atividades/tree 65,0-1 Bot
```

Figura 12: Explicação detalhada do script **tree**

```
> ./arqlab1/atividades/tree
+/home/nuke/faculdade/ea872/01-shell/files
+arqlab1
+atividades
- classifica
- folheto
- lixo
- menu
- path
- repeat_command.sh
- subspar
- test.sh
- traps
- tree
+exemplos
- diario
- prog1
- prog2
- prog3
- prog4
- prog5
- prog6
- teste1
- teste2
~/faculdade/ea872/01-shell/files main 13 ?3 12:23:22
>
```

Figura 13: Execução do script **tree**

Questão 10

O script está em anexo junto a este relatório. Abaixo há uma imagem ilustrando o seu funcionamento:

```
> ./repeat_command.sh --repeticoes=1 --atraso=3 echo "teste"
teste
> ./repeat_command.sh --repeticoes=3 --atraso=3 echo "teste"
teste
teste
teste
~/faculdade/ea872/01-shell/files/arqlab1/atividades main !3 ?3 9s 12:31:36
> █
```

Figura 14: Execução do script `repeat_command`. Observe a duração de 9s (3 execuções * delay de 3s).

Questão 11

Esse script também está em anexo junto ao relatório, e abaixo segue uma imagem dele em funcionamento:

```
> ./factorial.sh 5
120
> ./factorial.sh 20
2432902008176640000
> ./factorial.sh 21
Max supported factorial is 20! Aborting...
> ./factorial.sh -23
Invalid input: -23 (must be a non-negative integer)
> sleep
sleep: missing operand
Try 'sleep --help' for more information.
> ./factorial.sh 23 12
Usage: factorial.sh <number>
~/f/ea872/01-shell/files/arqlab1/atividades main !3 ?3 13:04:26
> █
```

Figura 15: Execução do script `factorial.sh`