13. Functions

[illegible] '[illegible]' [illegible] ([illegible]) [illegible]

```
. ./library.sh
```

[illegible]

[illegible]

[illegible]:

- [illegible]
- [illegible]
- **return** [illegible]
- **echo** [illegible] **stdout**[illegible], [illegible] c=`expr $a + $b`[illegible]

[illegible] C[illegible]. [illegible]

[illegible]:

```
#!/bin/sh
# A simple script with a function...


add_a_user()
{
USER=$1
PASSWORD=$2
shift; shift;
# Having shifted twice, the rest is now comments ...
COMMENTS=$@
echo "Adding user $USER ..."
echo useradd -c "$COMMENTS" $USER
echo passwd $USER $PASSWORD
echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}


###
# Main body of script starts here
###
echo "Start of script..."
add_a_user bob letmein Bob Holness the presenter
add_a_user fred badpassword Fred Durst the singer
add_a_user bilko worsepassword Sgt. Bilko the role model
echo "End of script..."
```

4[illegible] ()[illegible] [illegible] [illegible] [illegible] [illegible] . [illegible] [illegible] {[illegible] [illegible] , [illegible] } [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] . [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] . [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] .

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] echo[illegible] [illegible] , [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] . [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] !

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] . [illegible] [illegible] [illegible] [illegible] . [illegible] [illegible] add_a_user [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] . [illegible] " Start of script..."[illegible] echo [illegible] [illegible] . [illegible] [illegible] add_a_user bob letmein Bob Holness[illegible] [illegible] [illegible] [illegible] add_a_user [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] :

```
$1=bob
$2=letmein
$3=Bob
$4=Holness
$5=the
$6=presenter
```

[illegible] [illegible] [illegible] [illegible] $1[illegible] [illegible] [illegible] $1[illegible] [illegible] [illegible] [illegible] [illegible] bob[illegible] [illegible] . [illegible] [illegible] [illegible] '[illegible] ' $1[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] : A=$1[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] . [illegible] [illegible] [illegible] [illegible] $A[illegible] [illegible] [illegible] [illegible] . [illegible] [illegible] [illegible] [illegible] $3 [illegible] [illegible] $@[illegible] [illegible] . [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] . [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] .

??? ??

다른 언어에 익숙한 프로그래머라면 셸 함수의 범위 규칙에 놀랄 수도 있습니다. 기본적으로 매개변수 ($1, $2, $@ 등)를 제외하고는 범위 지정이 없습니다. 다음의 간단한 코드 세그먼트를 예로 들어보겠습니다:

```
#!/bin/sh

myfunc()
{
  echo "I was called as : $@"
  x=2 }

### Main script starts here

echo "Script was called with $@"
x=1
echo "x is $x"
myfunc 1 2 3
echo "x is $x"
```

이 스크립트를 scope.sh a b c로 호출하면 다음과 같은 출력을 제공합니다:

```
Script was called with a b c
x is 1
I was called as : 1 2 3
x is 2
```

함수 내에서 $@ 매개변수는 함수가 호출된 방식을 반영하기 위해 변경됩니다. 그러나 변수 x는 사실상 전역 변수 (global)로, myfunc가 이를 변경했으며 제어가 메인 스크립트로 돌아갈 때에도 이 변경 사항은 여전히 유효합니다.

함수의 출력이 다른 곳으로 파이프되는 경우 함수는 하위 셸에서 호출됩니다. 즉, "myfunc 1 2 3 | tee out.log"는 두 번째에도 여전히 "x는 1"이라고 표시됩니다. 이는 새로운 셸 프로세스가 호출되어 myfunc()를 파이프하기 때문입니다. 이것은 디버깅을 매우 불편하게 만들 수 있습니다. Astrid는 "| tee"가 추가되었을 때 갑자기 실패하는 스크립트가 있었는데, 왜 그래야 하는지 즉시 알 수 없었습니다. 간단한 예로 "ls | grep foo"의 경우, grep이 먼저 시작되어야 하고, ls가 시작되면 그 stdin이 ls의 stdout에 연결되어야 합니다. 셸 스크립트에서는 tee를 통해 파이프할 것이라는 사실을 알기도 전에 셸이 이미 시작되었으므로 운영 체제에서 tee를 시작한 다음 새 셸을 시작하여 myfunc()를 호출해야 합니다. 이것은 실망스럽지만 알아둘 가치가 있습니다.

함수는 호출된 값도 변경할 수 없으며, 스크립트에 전달된 매개 변수가 아닌 변수 자체를 변경해야 합니다. 예시를 통해 이를 보다 명확하게 확인할 수 있습니다:

```
#!/bin/sh

myfunc()
{
  echo "\$1 is $1"
  echo "\$2 is $2"
  # cannot change $1 - we'd have to say:
  # 1="Goodbye Cruel"
  # which is not a valid syntax. However, we can # change $a:
  a="Goodbye Cruel"
}

### Main script starts here

a=Hello
b=World
myfunc $a $b
echo "a is $a"
echo "b is $b"
```

이 다소 냉소적인 함수는 $a를 변경하여 "Hello World"라는 메시지를 "Goodbye Cruel World"로 바꿉니다 .

??(Recursion)

함수는 재귀적일 수 있습니다 . 다음은 팩토리얼 함수의 간단한 예입니다 :

```
#!/bin/sh

factorial()
{
  if [ "$1" -gt "1" ]; then
    i=`expr $1 - 1`
    j=`factorial $i`
    k=`expr $1 \* $j`
    echo $k
  else
    echo 1
  fi
}
```

```
while :
do
  echo "Enter a number:"
  read x
  factorial $x
done
```

약속한 대로 이제 셸 스크립트 간에 라이브러리를 사용하는 방법에 대해 간략하게 설명하겠습니다 . 나중에 살펴보겠지만 공통 변수를 정의하는 데에도 사용할 수 있습니다 .

common.lib

```
# common.lib
# Note no #!/bin/sh as this should not spawn
# an extra shell. It's not the end of the world # to have one, but clearer not to.
#
STD_MSG="About to rename some files..."

rename()
{
  # expects to be called as: rename .txt .bak
  FROM=$1
  TO=$2

  for i in *$FROM
  do
    j=`basename $i $FROM`
    mv $i ${j}$TO
  done
}
```

function2.sh

```
#!/bin/sh
# function2.sh
. ./common.lib
echo $STD_MSG
rename txt bak
```

function3.sh

```
#!/bin/sh
# function3.sh
. ./common.lib
echo $STD_MSG
rename html html-bak
```

[illegible] function2.sh[illegible] function3.sh[illegible] common.lib[illegible]. [illegible].

Exit Codes

[illegible] (14[illegible])[illegible]. [illegible].

```
#!/bin/sh

adduser()
{
  USER=$1
  PASSWORD=$2
  shift ; shift
  COMMENTS=$@
  useradd -c "${COMMENTS}" $USER
  if [ "$?" -ne "0" ]; then
    echo "Useradd failed"
    return 1
  fi
  passwd $USER $PASSWORD
  if [ "$?" -ne "0" ]; then
    echo "Setting password failed"
    return 2
  fi
  echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}

## Main script starts here
```

```
adduser bob letmein Bob Holness from Blockbusters
if [ "$?" -eq "1" ]; then
  echo "Something went wrong with useradd"
elif [ "$?" -eq "2" ]; then
   echo "Something went wrong with passwd"
else
  echo "Bob Holness added to the system."
fi
```

□ □□□□□ □ □□ □□ □□ (useradd □ passwd)□ □□□□ □□□ □□ □□□□□ □□□□□ . □□ □□ □ □□□ □□□ □□□ □□□ □□ □□ □□ □□ 1□ , □□□□□ □□□ □□ □□ □□ □□ 2□ □□□□□ . □□□ □□ □□ □□□□□ □□□ □□□ □□□ □ □ □□□□ .

Revision #1
Created 17 January 2024 01:23:46 by Enigma
Updated 17 January 2024 01:41:46 by Enigma