Functions

Part I

Functions

Part II

What You Will Learn

- Why to use functions
- How to create them
- How to use them
- Variable scope
- Function Parameters
- Exit statuses and return codes.

Why use functions? (Keep it DRY!)

- Don't repeat yourself! Don't repeat yourself!
- Write once, use many times.
- Reduces script length.
- Single place to edit and troubleshoot.
- Easier to maintain.

Functions

- If you're repeating yourself, use a function.
- Reusable code.
- Must be defined before use.
- Has parameter support.

Creating a function

```
function function-name() {
    # Code goes here.
}

function-name() {
    # Code goes here.
}
```

Calling a function

```
#!/bin/bash
function hello() {
    echo "Hello!"
}
hello
```

Functions can call other functions.

```
#!/bin/bash
function hello() {
    echo "Hello!"
    now
}
function now() {
    echo "It's $(date +%r)"
}
hello
```

Do NOT do this...

```
#!/bin/bash
function hello() {
    echo "Hello!"
    now
}
hello
function now() {
    echo "It's $(date +%r)"
}
```

Positional Parameters

- Functions can accept parameters.
- The first parameter is stored in $1.
- The second parameter is stored in $2, etc.
- $@ contains all of the parameters.
- Just like shell scripts.
  - $0 = the script itself, not function name.

Positional Parameters

```
#!/bin/bash
function hello() {
   echo "Hello $1"
}
hello Jason
# Output is:
# Hello Jason
```

```
#!/bin/bash
function hello() {
    for NAME in $@
    do
        echo "Hello $NAME"
    done
}
hello Jason Dan Ryan
```

Variable Scope

- By default, variables are global
- Variables have to be defined before used.

Variable Scope

```
GLOBAL_VAR=1
# GLOBAL_VAR is available
# in the function.
my_function
```

Variable Scope

```
# GLOBAL_VAR is NOT available
# in the function.
my_function
GLOBAL_VAR=1
```

```
#!/bin/bash
my_function() {
    GLOBAL_VAR=1
}
# GLOBAL_VAR not available yet.
echo $GLOBAL_VAR
my_function
# GLOBAL_VAR is NOW available.
echo $GLOBAL_VAR
```

Local Variables

- Can only be accessed within the function.
- Create using the **`local`** keyword.
  - local LOCAL_VAR=1
- Only functions can have local variables.
- Best practice to keep variables local in functions.

Exit Status (Return Codes)

- Functions have an exit status
- Explicitly
  - return <RETURN_CODE>
- Implicity
  - The exit status of the last command executed in the function

Exit Status (Return Codes)

- Valid exit codes range from 0 to 255
- 0 = success
- $? = the exit status

```
my_function
echo $?
```

```
function backup_file () {
  if [ -f $1 ]
  then
    BACK="/tmp/$(basename ${1}).$(date +%F).$$"
    echo "Backing up $1 to ${BACK}"
    cp $1 $BACK
  fi
}
backup_file /etc/hosts
if [ $? -eq 0 ]
then
  echo "Backup succeeded!"
fi
```

```
function backup_file () {
  if [ -f $1 ]
  then
    local BACK="/tmp/$(basename ${1}).$(date +%
F).$$"
    echo "Backing up $1 to ${BACK}"
    # The exit status of the function will
    # be the exit status of the cp command.
    cp $1 $BACK
  else
    # The file does not exist.
    return 1
  fi
}
```

```
backup_file /etc/hosts

# Make a decision based on the exit status.
if [ $? -eq 0 ]
then
  echo "Backup succeeded!"
else
  echo "Backup failed!"
  # About the script and return a non-zero exit
status.
  exit 1
fi
```

Summary

- DRY
- Global and local variables
- Parameters
- Exit statuses