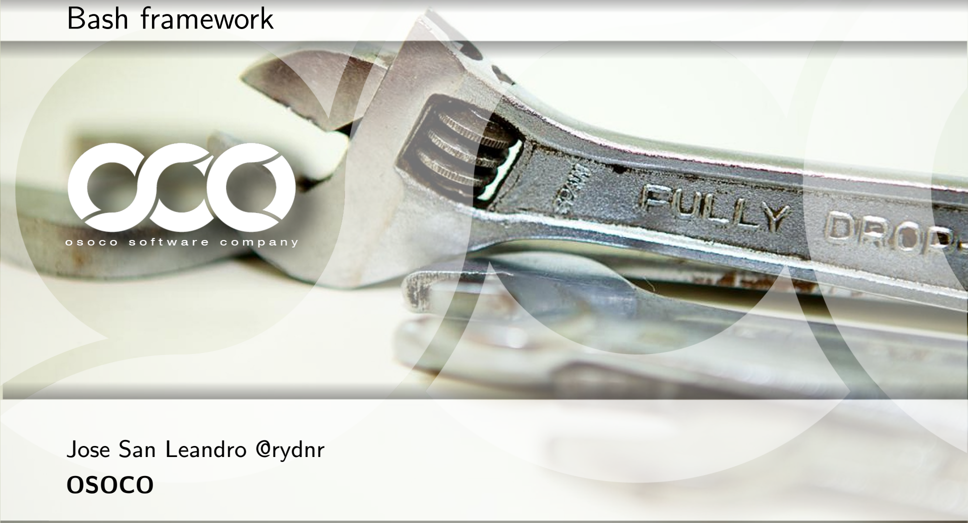
dry-wit

Bash framework



Jose San Leandro @rydnr
OSOCO

Contents

1 Introduction

2 dry-wit approach

3 dry-wit hooks

4 dry-wit API

Bash

In theory

- Powerful language.
- Conventions improve reusability and maintainability.
- Great fit for Unix and OSs with strong command-line focus.

Bash

In theory

- Powerful language.
- Conventions improve reusability and maintainability.
- Great fit for Unix and OSs with strong command-line focus.

In practice

- Easy for newcomers.
- Difficult to master.
- Write once, use twice, dispose.
- Expensive maintenance.
- Hardly reusable.

dry-wit: Features

Features

1. Provides a consistent way to structure scripts.
2. Manages required dependencies.
3. Declarative approach for dealing with errors (Output message + error code).
4. Handles parameters, flags, and environment variables.
5. API for creating temporary files, logging, etc.

Usage / Help

usage()

```
function usage() {
cat <<EOF
Does this and that...
Where:
  * myflag: changes the behavior somehow.
  * param1: the first input.
Common flags:
    * -h | --help: Display this message.
    * -v: Increase the verbosity.
    * -vv: Increase the verbosity further.
    * -q | --quiet: Be silent.
EOF
}
```

Input: checking

checkInput()

```
function checkInput() {
  local _flags=$(extractFlags $@);
  logDebug -n "Checking input";
  for _flag in ${_flags}; do
    case ${_flag} in
      -h | --help | -v | -vv | -f | --my-flag)
         ;;
      *) logDebugResult FAILURE "fail";
         exitWithErrorCode INVALID_OPTION ${_flag};
         ;;
    esac
  done
}
```

Input: parsing

parseInput()

```
function parseInput() {
  local _flags=$(extractFlags $@);
  for _flag in ${_flags}; do
    case ${_flag} in
      -f | --my-flag)
        shift;
        export MY_FLAG="${1}";
        shift;
        ;;
      *) shift;
        ;;
    esac
  done
}
```

Script requirements

Dependencies

```
function checkRequirements() {
  checkReq docker DOCKER_NOT_INSTALLED;
  checkReq realpath REALPATH_NOT_INSTALLED;
  checkReq envsubst ENVSUBST_NOT_INSTALLED;
}
```

DSL

1. *executable-file*: The required dependency.
2. `message`: The name of a constant describing the error to display should the dependency is not present.
3. dry-wit checks each dependency and exits if the check fails.

Error messages

defineErrors()

```
function defineErrors() {
  export INVALID_OPTION="Unrecognized option";
  export DOCKER_NOT_INSTALLED="docker is not installed. See http://www.docker.org";
  export REPOSITORY_IS_MANDATORY="The repository argument is mandatory";

  ERROR_MESSAGES=(\
    INVALID_OPTION \
    DOCKER_NOT_INSTALLED \
    REPOSITORY_IS_MANDATORY \
  );
  export ERROR_MESSAGES;
}
```

dry-wit takes care of the exit codes

```
exitWithErrorCode REPOSITORY_IS_MANDATORY;
```

main()

main()

```
function main() {
    // Focus on the logic itself.

    // Forget about defensive programming
    // regarding input variables
    // or dependencies.

    // Start by writing pseudo-code,
    // and later define the identified
    // functions.
}
```

Logging

logging

```
logInfo -n "Calculating next prime number ...";
...
if [ $? -eq 0 ]; then
  logInfoResult SUCCESS "done";
else
  logInfoResult FAILURE "Too optimistic";
fi
```

output

```
[2015/10/26 15:09:54 my-script:main] Calculating next prime number ...                [done]
```

Temporary files

API functions

1. Functions to create temporary files or folders.
2. dry-wit takes cares of cleaning them up afterwards.

createTempFile()

```
createTempFile;
local tempFile=${RESULT};
```

createTempFolder()

```
createTempFolder;
local tempFolder=${RESULT};
```

Environment variables: Declaration (1/3)

DSL for declaring environment variables

1. Declared in **[script].inc.sh**.
2. Safe to add to version control systems.
3. Mandatory information: description and default value.

defineEnvVar()

```
defineEnvVar \
    MYPASSWORD \
    "The description of MYPASSWORD" \
    "secret";
```

Environment variables: Overridding (2/3)

DSL for overridding default values

1. Declared in **.[script].inc.sh**.
2. Not always safe to add to version control systems.

overrideEnvVar()

```
overrideEnvVar MYPASSWORD ".toor!";
```

Environment variables: Overridding (3/3)

Environment value overridden from the command line

```
MYPASSWORD="abc123" ./script.sh ...
```