

PS/CO/Note. 90-08

CODE MANAGEMENT ON ULTRIX WORKSTATION

Franck Di Maio

Author: Franck Di Maio

Project: WSC

Revision: 1.2

Date: 10/17/90

File : code_management.me

CODE MANAGEMENT ON ULTRIX STATIONS

1. Environment classification

1.1. User environment

The user environment is the set of directories which are left to the responsibility of individual users. They are owned by individual account and project account used for program development.

Ex: /user1/lewis, /user1/oracle, /user2/wsc ...

1.2. Development environment

The development environment is the set of additions to the basic Ultrix environment which must be shared by a community of application developers.

It consists of:

- tools
- libraries
- include files
- manual pages
- documentations

Ex: /usr/local/include, /usr/local/bin ...

1.3. Operation environment

The operation environment is the environment for operational workstations.

1.4. Tester environment

The tester environment is the environment provided to external users in order to test (means try, not debug) programs or libraries which are being developed in the user environment.

This is a sub-part of the user environment which provides more stability than the development directories.

Ex: "tester" account, /user2/wsc/lib ...

Requirements

2.1. Operation environment

- (1) The operation environment must be independent from the other environment. These other environments may not be present on operational workstations. Ex: no /user1 nor /usr/local/bin on any operational ws.
- (2) The operational environment does not include central computers: priam or cemvm.
- (3) The operation environment is centrally managed. An installation procedure must be used to install software from the user environment into the operation environment.
- (4) It must be possible to identify to which package a file or symbol belongs to.

- (5) It must be possible to locate source file, author and version of any package.
- (6) Software for the operation environment must be delivered in source format.
- (7) Version management must be used (scs) in source and objects. The versions are archived as part of the installation procedure.
- (8) The installation procedure must include the generation of the code.
- (9) The generation procedure of the application must be delivered in a standard format (like a single "make" file). It must be valid for all targets (like Vax or risc).
- (10) Installation procedure must include on-line information (like "About" button) and documentation.
- (11) At run-time, all the executables and data files must be located in well known directories inside the operation environment. Ex: /psop/bin, /psop/uidlib , /psop/help ...
- (12) A system manager must be able to uninstall a version, (i.e. install a previous version) or to re-install an application from scratch. (remark: this means for instance that if the application needs a sub-directory, the creation of this sub-directory must be included in the generation procedure).
- (13) The installation of a new release is centrally managed. This means that the developers cannot install/fix themselves the soft in the development environment (??).

2.2. Development environment

- (1) A unique identifier must be allocated for each software package. This identifier must be used for sub-directories, external names ...
- (2) There must be no connections between the development environment and the user environment. Ex: the include files must be copied from a project account into /usr/local/include instead of establishing symbolic links.
- (3) Version numbers are centrally managed.
- (4) The sources for the software which application programs depend on is managed centrally. This include all the libraries and the commands which are used by applications. This may not include tools used in the development environment only.
- (5) A system manager must be able to uninstall a version.
- (6) Everything that a package exports: symbols, file paths, documentation... must be managed.

2.3. User environment

- (1) The only constraints existing in the user environment are the one related to the system management: centralized allocation of user names and ids, disk quota...

3. Preliminary scenario An unique identifier is allocated for each application. This identifier is used for sub-directories, scs id, data files ...

3.1. Operation environment

The file system of an operational workstation is composed of the following parts:

- The "system" directories, common to every Ultrix WS: (/usr and /).
- The "operation" directory which contains every files which need to be read during operation and which are only modified during the installation of an application: object files, scripts, uid files, documentation...
- The "temp" directory (/tmp) where any volatile data may be written/read by the applications. Volatile means necessary only during the execution of a program. Files in this directory will be purged according to the last time read (ex: > 24h) and everytime the user logs in (??).
- The "data" directory which contains every file which need to be written/read during operation and which (1) needs to be maintained or (2) needs to be modified externally to the operational environment.

3.1.1. Operation directory

The operation directory is duplicated on 2 servers: one for Vax and one for Risc. Only a privileged user ("installer") as write access in it. It is mounted on all operational workstations as well as on development workstation if necessary.

Structure:

/psop	
bin	executables
uidlib	uid files
helplib	help directories
doc	documentation directories (postscript files)

3.1.2. Data directory

The data directory is unique for the whole network. It can be mounted on any "supported" workstation (a managed list).

The owner of all sub-directories is a privileged user ("installer"). For each new data set, a dedicated group must be allocated. Write access to the data is granted via group permissions. Therefore a list of authorized users must be maintained for each data set by means of the group definition.

Example: data files: "oplog/monday," "oplog/tuesday" ...

directory: /data/oplog, owner = "psinst", group = "oplog"

file access: owner = rw, group = rw, others = r

oplog group members: psinst,psop

files: /data/oplog/monday, /data/oplog/tuesday ..., owner = "psop", group = "oplog"

The data directory is daily backed-up. The global size is limited and quota are used to manage the amount of space allocated to each user.

3.2. Development environment

The file system of an development workstation is composed of the following parts:

- The "system" directories, common to every Ultrix WS: (/usr and /). this may include "un-supported" software, like emacs, rand...
- The "local stuff" directory which contains local additions, like libraries, tools...
- The data directory if needed.

3.2.1. Local stuff directory

The local directory must include all additions made to the base system. It is duplicated on 2 servers: one for Vax and one for Risc. Only a privileged user ("installer") as write access in it. It is mounted on all development workstations.

Structure:

/usr/local	
bin	executables
lib	libraries
include	include files
man1	man pages sources (link from /usr/man/man1)
cat1	man pages formatted (link from /usr/man/cat1)

3.3. Software installation

Installations into either the operation environment or the development environment need to be managed centrally.

Every piece of code developed internally must be installed into a dedicated directory and must satisfy to a set of standards.

3.3.1. Installation directory

The installation directory must be unique in the whole network. Only a privileged user ("installer") as write access in it.

This directory must contain source code only, object code is generated inside this directory and cleaned after installation.

Structure:

```
/psinst
    prog      application directories
    lib       libraries
    util      generation utilities
    Makefile  whole installation, start make in subdirectories
```

Application directory (ex: codd):

```
/psinst/prog/codd
    Makefile  commands for generation, installation, cleaning ...
    SCCS      *.c, *.h, *.uil, help/* doc/* ...
```

3.4. Standards

Each package to be installed must comply to a set of standards in order to prevent collisions and to allow centralized maintenance.

3.4.1. Library conventions

When installing a library the following standards should be applied. In the examples, the library identifier is "stuff".

- Exported include file names are in lower case, words being separated by "_", the first word being the package identifier. Ex: stuff.h or stuff_part1.h, stuff_part2.h.
- Internal include files (needed only for generating the library or associated tools) should used the same syntax plus a capital "P". Ex: stuffP.h, stuff_part1P.h ...
- Every constant, type, procedure or variable exported by the package must be declared in the exported include file(s).
- The following naming conventions (from Motif) should be applied:
 - Exported constants (#define): StuffTHE_CONSTANT (applies to members of enum also)
 - Exported types: StuffTheStructure
 - Exported routine names: StuffDoThis ().
 - Fields of exported records: first_field (lower case separated by underscore).
 - Exported variables: stuffTheVariable
- Exported include files should include conditionals to prevent duplicated inclusions (a la Motif), symbols should be the same as file names in upper case.

```
#ifndef STUFF
#define STUFF
-- all statements here
#endif
```
- Common statements should be grouped into include files shared by all users of the development environment. This concerns statements already "somewhere" in the system or new local statements Ex: "NULL", "Boolean", "String", "min", etc. plus "WordArray", "LongArray", etc.
- All the library code must be put into a library file whose name is built from the identifier. Ex: libstuff.a
- A sccs variable must be generated in the main file (stuff.c) with the "@(#)code_management.me 1.2 10/17/90" pattern. Ex: static char stuff_sid [] = "@(#)code_management.me 1.2 10/17/90";

- Tool files (compiled or scripts) should follow the same convention as include files, source files also. Ex: stuff_tool (executable), stuff_tool.c (source). A scs variable must also be generated for each tools.
- Manual pages must be provided documenting the package.
- Every symbol defined in an exported include file must be referred to in a man page.
- Each tool must be referred to in a man page.
- Test program(s) must be provided.

3.4.2. Application conventions

Same problem.

- All path names must be built using an environment variable (exceptions: temporary files and Ultrix files). This implies that no absolute path is used for the following files, but the path is built from an environment variable and a file name:
 - executables use PATH
 - uid files use UIDPATH
 - help files use HELPPATH
 - data files use DATAPATH
 - etc.
- The main file name must be the same as the identifier of the application (ex: codd.c).
- A scs variable must be generated in the main file with the "@(#)code_management.me 1.2 10/17/90" pattern.

3.5. Installation procedures

3.5.1. Installation of a library (stuff)

- log in the installer account
- copy the new version of the sources into ~/lib/stuff
- archive the new sources,
- generate the new version,
- check for application dependencies (how?),
- update and generate the test program,
- run the test program,
- install into the development environment:
 - install libstuff.a into /usr/local/lib (+ ranlib)
 - copy include(s) into /usr/local/include
 - copy man page(s) into /usr/local/man1
 - etc.
- log the installation ?.

3.5.2. Installation of an application (codd)

- log in the installer account
- copy the new version of the sources into ~/prog/co
- archive the new sources,
- generate the new version,
- test the version in restricted environment (???),
- install into the operation environment:

copy executable into /psop/bin/codd
copy uids into /psop/uidlib/codd.uid or /psop/uidlib/codd/*.uid
etc.

log the installation ?.