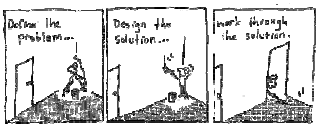
SYST13416
Introduction to Linux
Operating

BASH Shell Scripting

PROGRAM DEVELOPMENT CYCLE

Objectives

- Understand the importance of documentation
- Learn to document well
- Inline (comments) documentation
- Manual pages



Program Development Cycle

The process for developing an application involves the same steps when

- Writing new high-level language programs (Java, VB, C/C++, C#, Javascript, …)
- Fixing an existing problem or developing a new application
- Writing shell scripts

Example

Write out all steps to complete the following task:

a user-friendly version of ”Copy A File”

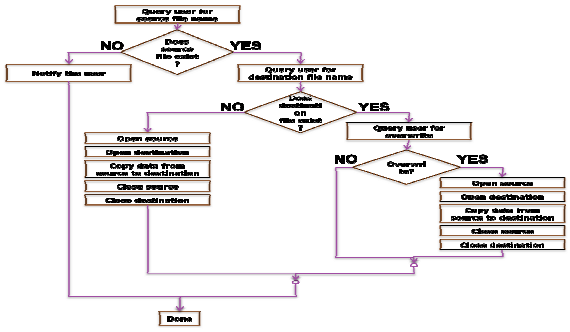
Pseudocode

1 Query the user for the name of the source file
2 Check whether the source file exists
3 If the source file doesn’t exist, notify the user
4 Query the user for the name of the destination file
5 Check whether the destination file exists
6 Open the source file
7 Open the destination file
8 Copy the data from the source file to the destination file
9 Close the source file
10 Close the destination file

- Note that each step should be described with a complete natural language, which includes at minimum a verb (action) and a noun. (Modifiers are optional)
- A well defined set of circumstances that can be described using a sufficiently complete natural language. A sufficiently complete natural language will include at least a noun, a verb, and some modifiers.
- A noun is the name of a type of person, animal, plant, place, thing, substance or idea. A proper noun is the name of a particular occurrence or instance of a noun. A pronoun is a word used as a substitute for a noun and that refers to a noun named or understood in the context in which it is used
- A verb is a word that describes a mode of being, an association, an action or an event. Verbs describe the state of nouns, and relate the nouns of a situation with one another. Verbs can be either active or passive
- A modifier is a word that qualifies a noun or a verb regarding its character, quantity, degree or extent. Modifiers of nouns are called adjectives, while modifiers of verbs are called adverbs.

Pseudocode

1 Query the user for the name of the source file
2 Check whether the source file exists
3 If the source file doesn't exist, notify the user
4 Query the user for the name of the destination file
5 Check whether the destination file exists
6 Open the source file
7 Open the destination file
8 Copy the data from the source file to the destination file
9 Close the source file
10 Close the destination file



Pseudocode

1 Query the user for the name of the source file
2 Check whether the source file exists
3 If the source file doesn't exist, notify the user
4 Query the user for the name of the destination file
5 Check whether the destination file exists
6 If the destination file exists, ask the user if he want to replace it
7 Open the source file
8 Inform the user if the source file is empty. If so, exit.
9 Open the destination file
10 Copy the data from the source file to the destination file
11 Close the source file
12 Close the destination file

```
#!/bin/bash
# add standard header: who, what, where, why, when

# Query the user for the name of the source file
echo –n "Enter the name of the source file, ending with return: "
read source

# Check whether the source file exists
if [ -f $source ]; then

# Query the user for the name of the destination file
    echo –n "Enter the name of the destination file, ending with return: "
    read destination
else
# If the source file doesn't exist, notify the user
    echo "The source file does not exist. No action taken."

# Check whether the destination file exists
# If the destination file exists, ask the user if he want to replace it

# Open both files, copy the data from the source file to the destination file, and close both files
    cp $source $destination
fi
```


SYST13416 Introduction to Linux Operating

Summary of the English Language

- ***Situation***
  A well defined set of circumstances that can be described using a sufficiently complete natural language. A sufficiently complete natural language will include at least the following *three grammatical constructs*.
- ***Noun***
  A ***noun*** is the name of a type of person, animal, plant, place, thing, substance or idea. A ***proper noun*** is the name of a particular occurrence or instance of a noun. A ***pronoun*** is a word used as a substitute for a noun that refers to a noun named or understood in the context in which it is used.
- Nouns can represent animate or inanimate objects, and the objects may be either tangible or intangible. It is impossible to describe any situation without the use of at least one noun.
- ***Verb***
  A verb is a word that describes a mode of being, an association, an action or an event. Verbs describe the state of nouns, and relate the nouns of a situation with one another. Verbs can be either active or passive.
- ***Modifiers***
  A modifier is a word that qualifies a noun or a verb regarding its character, quantity, degree or extent. Modifiers of nouns are called *adjectives*, while modifiers of verbs are called *adverbs*.
- Situations that can be described without the use of modifiers are usually too trivial to be of interest, though they do exist.

SYST13416
Introduction to Linux
Operating

Advanced Commands

DOCUMENTATION AND TESTING

Objectives

- Create and incorporate manual pages for your own commands and utilities
- How do you know that your program is functioning according to the specifications?
- How do you know that your logic is correct?
- How do you know that the results will be correct under all conditions?
- Does your program handle all errors?

Documentation

Inline (comments)

Manual Pages

Why is documentation so important?

- DOCUMENTATION is a common language with others who may be working with us directly or are involved in the Customer's environment or on the same project.
- Documentation also enables better maintenance of the solution over time and implementation of any CHANGES when business conditions demand it.
- Sometimes, documentation must be created by us. Other times, it is created by others and we need to be able to understand it, so that we can act upon it appropriately and incorporate the information correctly in our steps of the development process.
- There are techniques and methods that have become standard over the last few decades in the industry to document both the PROBLEM and the SOLUTION.

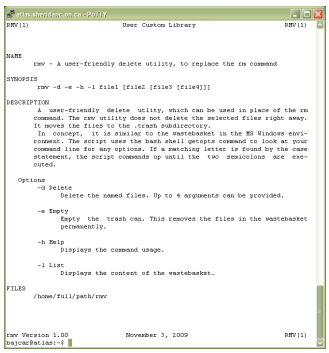
groff COMMAND

Formatting tags:
The .**TH tag** formats the page title
The **.SH tag** indicates a section.
The **.SS tag** indicates the beginning of a subsection.
The **.TP tag** indicates each item in the Options subsection.
The **\fB tag** changes the font to boldface.
The **\fI tag** changes the font to italic.
The **\fR tag** changes the font to roman.
The **\fP tag** changes the font to its former setting.

WRITE AND FORMAT YOUR OWN MANUAL PAGE

- On a server, you would include manual pages in the same directory as the existing manual pages. In your own space, create your manual pages in a directory where you would keep your executables, such as ~/bin directory. Recall that the ~ indicates your home directory.
- The manual pages have the extension .1 (one) and basename would be the same as your script or utility. For example, in an earlier unit, you created a script called rmv. To create a manual page for this utility, use the vi editor to create the file "rmv.1".
- Type the text on the following page into the file.
- Once you've created the file, save and exit vi.
- At the command line, test the man page by typing
  `groff –Tascii –man rmv.1`

  Your screen should appear similar to the standard man pages for other commands. Alternately, you can pipe the command to more to page through the document (press q to quit):
  `groff –Tascii –man rmv.1 | more`
- After running the gruff command, you can test your new man page by typing
  `man ./rmv.1`
- Note that since you are not the administrator, you are running it locally to your account and we use the ./ to indicate that the page can be found in your current directory, rather than the system one.
- There are six common sections are:
  NAME: the name of the command or program
  SYNOPSIS: a brief description of the command or program
  DESCRIPTION: a detailed description of the command or program
  FILES: a list of files used by the command or program
  SEE ALSO: a list of other commands or programs that are related to this one
  BUGS: a list of known bugs
- Section names usually appear in all uppercase characters on a man page.



- Contents of the rmv.1 file:

```
.TH RMV 1 "November 3, 2009" "rmv Version 1.00" "User Custom Library"
.SH NAME
rmv \- A user-friendly delete utility, to replace the rm command
.SH SYNOPSIS
\fB rmv –d –e –h –l file1 [file2 [file3 [file4]]]\fP
.SH DESCRIPTION
\fP A user-friendly delete utlity, which can be used in place of the
rm command. The rmv utility does not delete the selected files right
away. It moves the files to the .trash subdirectory. \fP.
\fP In concept, it is similar to the wastebasket in the MS Windows
environment. The script uses the bash shell getopts command to look
at your command line for any options. If a matching letter is found
by the case statement, the script commands up until the two
semicolons are executed.
\fP.
.SS Options
.TP
\fB –d \fIDelete\fR
Delete the named files. Up to 4 arguments can be provided.
.TP
\fB –e \fIEmpty \fR
Empty the trash can. This removes the files in the wastebasket
permanently.
.TP
\fB –h \fIHelp\fR
Displays the command usage.
.TP
\fB –l \fIList \fR
Displays the content of the wastebasket.
.SH FILES
.TP
\fC/home/full/path/rmv\fR
```

Sheridan

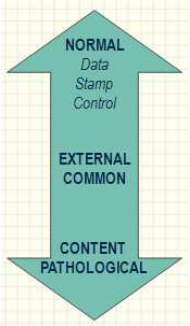
SYST13416
Introduction to Linux
Operating

Advanced Commands

FUNCTIONS: COHESION & COUPLING

Objectives

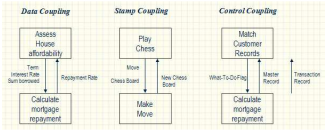
- Create and incorporate manual pages for your own commands and utilities.



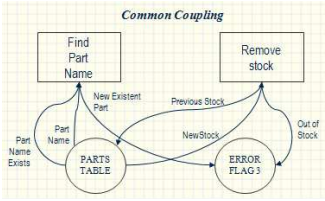
How do you know when to use a procedure?
How do you know what parts of your code to place in which procedures?
Do you group by number of statements?
Do you group by what the statements do?
Do you group by the data that is being used?
These questions are answered when we study coupling and cohesion

Coupling
The strength of the connection between two modules
How much a module depends on the data from other module
Tight and loose coupling
Many levels of coupling

Loose Coupling
Occurs when procedures do not depend on each other.
Values are passed from one module to another
Easier to re-use in different programs because they are not dependent upon modules in one specific program



Data Coupling
the loosest type of coupling
occurs when modules share data by passing arguments (or parameters)
Data-Structured Coupling or Stamp Coupling (fig. 14-8 pg. 401)
occurs when an entire record is passed to a module
easier than passing multiple fields if you need a lot of them from one record
Control Coupling (fig. 14-4 pg. 402)
occurs when a main module passes an argument to another module, controlling the module's actions
Control Coupling (fig. 14-4 pg. 402)
occurs when a main module passes an argument to another module, controlling the module's actions
this is a pretty tight type of coupling
the sub procedure depends on getting certain values
the calling program must know how to get a valid choice from the user



if you change the sub procedure, you have to change the calling procedure, too

Tight Coupling
Occurs when there is a lot of dependency between modules
modules all have access to the same globally defined variables
when one module changes a value, other modules are affected
More prone to errors and allows bad data to pass from one module to another
Data from one module could be altered unnecessarily by another
External Coupling: when two or more modules access the same global variable
Common Coupling: when two or more modules access the same global record
Changes to values and field definitions in one module affect other modules and they would have to be changed too

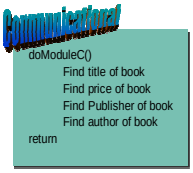
this is a pretty tight type of coupling
the sub procedure depends on getting certain values
the calling program must know how to get a valid choice from the user
if you change the sub procedure, you have to change the calling procedure, too

Content Coupling
Pathological Coupling
occurs when two or more procedures change one another's data
makes a program confusing and difficult to follow

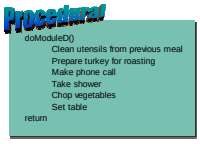
What type of coupling is indicated by each of the following interfaces?
Perform printLabelA(name, street, city, province, postal)
Perform printLabelB(custMailingAddress)
Perform printLabelC(custPurchOrderRecord)
Perform printLabelD()

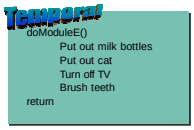
Cohesion
How the statements inside a module accomplish the objective or purpose of the module
Modules should be highly cohesive
all statements are related to each other
the statements all help to perform one task
makes programs easier to write, read and maintain

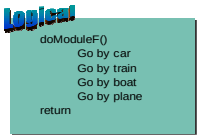
Functional Cohesion
Occurs when all the statements in a module contribute to the performance of a single task
The highest level of cohesion
NOT functionally cohesive
Calculate gross pay, figures out the tax deductions and prints a paycheck

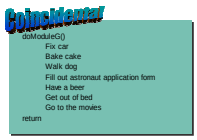


Functionally cohesive
Calculate the Celsius conversion of a Fahrenheit temperature
Compute cosine of an angle
Verify alphabetic syntax
Read transaction record
Determine mortgage repayment schedule
Compute point of impact of a missile
Calculate net employee salary
Assign seat to an airline customer









Sequential Cohesion
occurs when a procedure has a specific order of statements using the same data
A weaker form of cohesion
the module might perform several different tasks
statements are related because they work on the same data and seem like a series of steps
the order of the steps is important

Communicational Cohesion
Occurs in procedures that contain statements that perform tasks that share data
Similar to sequential cohesion, but the tasks do not need to be in any particular order
A weaker form of sequential cohesion
Examples
Checking for specific values in a variable and performing tests for validation

Procedural Cohesion
when statements are done in sequence but with different data
main-line logic is an example

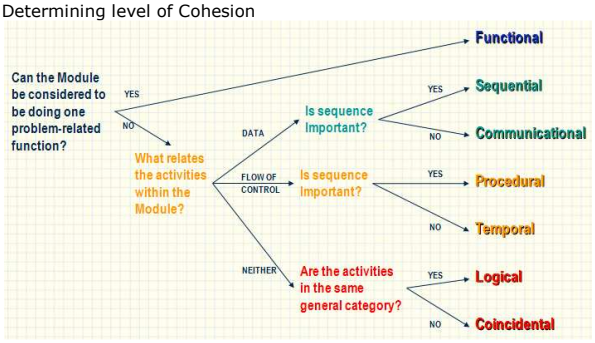
Temporal Cohesion
when statements in a procedure are related by time
they are placed together because of when they must occur, such as at the beginning of a pgm

Logical Cohesion
when a procedure contains tasks that are related and dependent on the results of selection structures
a record maintenance menu might be an example

Summary
- Reduce Coupling!
- Increase Cohesion!

Coincidental Cohesion
when a procedure contains statements that just happen to be together
weakest - this really isn't cohesive at all!

Determining level of Cohesion



Effects of Cohesion

Cohesion level	Coupling	Cleanliness of Implementation	Modifiability	Understandability	Effect on overall system maintenability
Functional	Good	Good	Good	Good	Good
Sequential	Good	Good	Good	Good	Fairly good
Communicational	Medium	Medium	Medium	Medium	Medium
Procedural	Variable	Poor	Variable	Variable	Bad
Temporal	Poor	Medium	Medium	Medium	Bad
Logical	Bad	Bad	Bad	Poor	Bad
Coincidental	Bad	Poor	Bad	Bad	Bad

Exercise
What Level of Cohesion are they?

```
checkCorrectness()
        check syntactic correctness of
            space-vehicle guidance parameters
return

generateReport()
        if condition is 'sales' then
            produce sales report
        else
            if condition is 'project' then
                produce project status report
            else
                produce customer transaction report
            end if
        end if
return

updateNewTransaction()
        update record on file
        get next transaction
return

doHousekeep()
        open files
        obtain first transaction
        obtain master record
        print page headings
return

updateCredit()
        update current credit record
        write it to CD
return
```

FUNCTIONAL – CheckCorrectness()
LOGICAL – generateReport()
PROCEDURAL – updateNewTransaction()
TEMPORAL – doHousekeep() -- these activities are done at one time
SEQUENCIAL – updateCredit()

Functions (complex alias)

Syntax

```
name() { commands }
function name { commands }
function name() { commands }
```

- Follow Unix name conventions
- Avoid renaming a current command or alias
- Surround both braces with white space
- Separate each command with appropriate separator
- Terminate the last command with ;
- Arguments that you pass to the function become positional parameters

Example

```
function cdp() { cd $1; echo $PWD; }
cdp someDir
```

Question

Create a function called cdl that changes a directory and lists its content.