CSCE 747 - Assignment 1:
Introduction and Functional Testing

Due Date: Thursday, February 2nd, 11:59 PM

There are 8 questions, worth a total of 100 points. You may discuss these problems in your teams and turn in a single submission for the team, in PDF format, on Moodle. Answers must be original and not copied from online sources.

Problem 1 (10 Points)

Answer Exercise 2.4 from the textbook. For each option, address how pessimistic/optimistic it is relative to the rest and how simple/complex it is to check relative to the rest, providing brief explanation for your rankings.

Problem 2 (10 Points)

Answer Exercise 3.1 from the textbook. Do not simply list a principle, but explain your reasoning.

Problem 3 (10 Points)

The following properties emerge during the development of the software for an Automatic Teller Machine (ATM). For each of the following, identify whether it is a correctness, robustness, or safety property. Briefly justify your decision.

1. If the network connection is interrupted, the session shall be immediately terminated with an appropriate error message. No funds shall be dispensed, and no changes shall be made to a user's account.

2. The account identifier of the account loaded by the utility shall exactly match the account identifier of the input debit card.

3. The amount requested by the user shall only be debited from their account following confirmation of the successful withdrawal of funds.

4. If no physical money remains in the unit following a completed transaction, the utility shall cease standard operation and display an error message until a manual override is initiated.

Problem 4 (10 Points)

Under what circumstances can making a system more safe also make it less reliable. Briefly explain with an example.

Problem 5 (15 Points)

Exercise 10.3 in the text book. Note: You must cover all the functionality stated in the specification.

Problem 6 (25 Points)

For the GNU tail utility (described at the end of this document), derive parameter characteristics, representative values, and semantic constraints suitable for generating a set of test case specifications using the category partition method. See page 187 in the textbook for an example solution to a similar problem discussed in the book.

Note – when a flag uses capital letters for input (for example, --pid=PID), that means that the user supplies a value. When lower-case is used (for example, --follow=name), that means the literal option (name).

Problem 7 (10 Points)

Calculate an upper bound on the number of tests required to cover all 2-way, 3-way and 4-way interactions for the tail utility. If there are multiple aliases for the same parameter characteristic (-n and –lines) and no functional difference between the two, you only need to assign those to one parameter (e.g., you don't need both -n and –lines). If there is a behavioral difference (-f and –follow don't always have the same effect), you should model all possibilities.

Problem 8 (10 Points)

Category-partition testing starts with an exhaustive approach and narrows down the final number of tests through the use of constraints. While some of these are based on restrictions in combinations of values (properties and if-properties), others are somewhat arbitrary decisions intended solely to limit the number of test cases (single, error).

When utilizing pairwise combination testing, constraints can also be imposed. These are constraints on what pairs of values are possible, and they come in two forms:
invalid pairs: when parameter A = X, parameter B must never = Y.
married pairs: when parameter A = X, parameter B must always = Y.

For the GNU tail example, derive constraints for invalid or married pairs (you may be able to derive some of these from the category-partition constraints you came up with for problem 6).

List the constraints and justify why these constraints are valid.

Tail Documentation

NAME

tail - output the last part of files

SYNOPSIS

tail [OPTION]... [FILE]...

DESCRIPTION

Print the last 10 lines of each FILE to standard output. With more than one FILE, precede each with a header giving the file name. With no FILE, or when FILE is -, read standard input.

Mandatory arguments to long options are mandatory for short options too.

-c, --bytes=K

output the last K bytes; or use -c +K to output bytes starting with the Kth of each file

-f, --follow[={name|descriptor}]

output appended data as the file grows;

an absent option argument means 'descriptor'

-F same as --follow=name --retry

-n, --lines=K

output the last K lines, instead of the last 10; or use -n +K to output starting with the Kth

--max-unchanged-stats=N

with --follow=name, reopen a FILE which has not

changed size after N (default 5) iterations to see if it has been unlinked or renamed (this is the usual case of rotated log files); with inotify, this option is rarely useful

--pid=PID

with -f, terminate after process ID, PID dies

-q, --quiet, --silent
never output headers giving file names

--retry
keep trying to open a file if it is inaccessible

-s, --sleep-interval=N
with -f, sleep for approximately N seconds (default 1.0) between iterations; with inotify and --pid=P, check process P at least once every N seconds

-v, --verbose
always output headers giving file names

--help display this help and exit

--version
output version information and exit

If the first character of K (the number of bytes or lines) is a '+', print beginning with the Kth item from the start of each file, otherwise, print the last K items in the file. K may have a multiplier suffix: b 512, kB 1000, K 1024, MB 1000*1000, M 1024*1024, GB 1000*1000*1000, G 1024*1024*1024, and so on for T, P, E, Z, Y.

With --follow (-f), tail defaults to following the file descriptor, which means that even if a tail'ed file is renamed, tail will continue to track its end. This default behavior is not desirable when you really want to track the actual name of the file, not the file descriptor (e.g., log rotation). Use --follow=name in that case. That causes tail to track the named file in a way that accommodates renaming, removal and creation.

AUTHOR

Written by Paul Rubin, David MacKenzie, Ian Lance Taylor, and Jim Meyering.

REPORTING BUGS

GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
Report tail translation bugs to <http://translationproject.org/team/>

COPYRIGHT

SEE ALSO

Full documentation at: <http://www.gnu.org/software/coreutils/tail>
or available locally via: info '(coreutils) tail invocation'