Auxilium User's Guide

Version 1.0.0

Contents

Auxilium Introduction

Auxilium is the latin word for *support*. The “auxilium” package is an open-source project which contains a collection of bash shell executable and sourceable scripts which support, and make easier, many common Unix/Linux user tasks. All auxilium utilities are written in *bash“, the standard Linux interpretive shell language. This facilitates portability (between bash platforms) and ease of installation. Auxilium has no runtime prerequisites other than bash version 4.2 or higher.

The core auxilium utilities include:

- **auxenv**. Setup, manipulation, and optimization of path-like environment variables such as *PATH*, *MANPATH*, and *LD_LIBRARY_PATH*.
- **auxwhere**. Find files by filename search through path-like environment variable directories.
- **auxsource**. An extended version of the bash built-in *source* command.
- **auxalias**. Creates static or dynamic aliases for all auxilium sourceable scripts.
- **auxchecktap**. Scans and checks TAP format unit test result logs, producing a summarized TAP test results log.
- **auxguid**. Generates a pseudo random 128-bit (32 hex character) GUID/UUID.
- **auxilium**. Displays version info for auxilium and all of its compoents.

Auxilium Installation

Install of auxilium runtime distribution

Runtime installation is relatively easy. As an example, below is a sample installation of auxilium 1.0.0, which is pretty much the same for other auxilium versions if you simply adjust the release version number.

1. Download the runtime distribution. There are at least two ways of doing this:

    a. Primary download method: Right-click on the “1.0.0” link on this page at https://ericmotleybytes.github.io/auxilium/ and select “save link as…”.

    b. Alternate download method: Use the “wget” program to get the tar file.

    ```
    $ wget https://github.com/ericmotleybytes/auxilium\
    /releases/download/1.0.0\
    /auxilium-run-1.0.0.tar.gz
    ```

    This will create the file “auxilium-run-1.0.0.tar.gz”.

2. Unpack the tar file.

    ```
    $ tar -xzf auxilium-run-1.0.0.tar.gz
    ```

    This will create a sub-directory called “auxilium-run-1.0.0” which contains additional files and directories.

3. Set the default directory to the “auxilium-run-1.0.0” sub-directory.

    ```
    $ cd auxilium-run-1.0.0
    ```

4. Use make to install auxilium under your desired directory prefix.

    a. Personal install: To install auxilium under a personal directory, for example under the ~/local directory, issue the following command:

    ```
    $ make install PREFIX=~/local
    ```

This installs executable scripts in ~/local/bin, man pages in ~/local/man/man1, html files in ~/local/share/html, and so forth.

   b. System install: To install auxilium under a shared system-wide directory, for example under the /usr/local directory, issue the following command:

      ```
      $ sudo make install PREFIX=/usr/local
      ```

5. Optionally, although this is highly recommended, setup automatic auxilium alias definitions in the appropriate login script.

   a. Personal setup: Edit your ~/.bashrc file and add the line:

      ```
      source ~/local/bin/auxalias
      ```

   b. System setup: Edit the /etc/bash/bashrc file and add the line:

      ```
      source /usr/local/bin/auxalias
      ```

Congratulations! You have now installed auxilium!

If your MANPATH is defined resonably you should be able to see an overview of auxilium by typing:

```
$ man auxilium
```

Uninstall of auxilium runtime distribution

You can also easily uninstall auxilium. As an example, below is a sample uninstall of auxilium 1.0.0, which is pretty much the same for other auxilium versions if you simply adjust the release version number.

1. Set the default directory to the “auxilium-run-1.0.0” sub-directory.

   ```
   $ cd auxilium-run-1.0.0
   ```

2. Use make to uninstall auxilium under your desired directory prefix.

   a. Personal uninstall: To uninstall auxilium from under a personal directory where it was previously installed, for example under the ~/local directory, issue the following command:

      ```
      $ make install PREFIX=~/local
      ```

      This remove only auxilium installed files.

   b. System install: To uninstall auxilium under a shared system-wide directory where it was previously installed, for example under the /usr/local directory, issue the following command:

      ```
      $ sudo make uninstall PREFIX=/usr/local
      ```

3. Optionally, if you edited a login script during installation, your might want to delete or comment out the line you added.

   a. Personal setup: Edit your ~/.bashrc file and remove or comment the line:

      ```
      source ~/local/bin/auxalias
      ```

      If commented, the line will then look like:

      ```
      #source ~/local/bin/auxalias
      ```

   b. System setup: Edit the /etc/bash/bashrc file and remove or comment the line:

      ```
      source /usr/local/bin/auxalias
      ```

      If commented, the line will then look like:

      ```
      #source /usr/local/bin/auxalias
      ```

4. Set the default directory to the original download directory and delete the tar distribution file and the unpacked files.

   ```
   $ cd <download-directory>
   $ rm auxilium-run-1.0.0.tar.gz
   $ rm -rf auxilium-run-1.0.0
   ```

You have now uninstalled auxilium.

Utilities

auxenv

auxenv - manipulate path-like environment variables

auxenv introduction

A sourceable script. Facilitates setup, manipulation, and optimization of path-like environment variables such as *PATH*, *MANPATH“, *LD_LIBRARY_PATH*, *PKG_CONFIG_PATH*, *PYTHONPATH*, and more. The auxenv utility is a valuable tool for both system administrators and end-users who have complex and/or multiple environment configuration issues.

auxenv usage

```
auxenv [<option>...] <command> <envname> [<arg>...]
  --ifs="<char>"            # define separator character
  --ifs=x<hex><hex>         # define sep char in hex
  --unset-empty             # unset env var when empty
  -h | --help               # display help
  -v | --version            # display version
```

(Assuming ‘auxenv’ is an alias to ‘source <dir>/auxenv’.)

auxenv options

–ifs="<char>" (default is ":") Specifies the character which separates items within the environment variable value. For example, on Linux directories within the PATH ennvironment variable are separated by a colon (":") character. A colon (":") is the default separator if this option is not specified.

–ifs=x<h><h> Same as the –ifs option above, except the separating character can be specified via its ASCII value in two hex digits. For example, –ifs=x20 specifies a space character.

–unset-empty If the final environment variable value being manipulated ends up being "" (an empty string) this option will cause the environment variable to be unset, not just left with a "" value.

-h | **–help** Displays a brief auxenv command usage summary. If used, this option must be the only option or parameter on the command line.

-v | **–version** Displays auxenv version information. If used, this option must be the only option or parameter on the command line.

auxenv commands

append <envvar> <item-or-dir> Modifies the value of the environment variable so that the item (usually a directory) is placed as the very last item of the new environment variable value.

clean <envvar> This is a convenient way to do commands "desym", "dedup", and "weed", in that order. See descriptions of those commands below for more information.

count <envvar> Parses the environment variable value into items (usually directories) and outputs a count of how many items were found.

dedup <envvar> [topdown|bottomup] Modifies the value of the environment variable by eliminating literal duplicates within listed items (usually directories). If 'topdown' is specified (also the default), then the first item match is kept. If 'bottomup' is specified, then the last item match is kept.

desym <envvar> Modifies the value of the environment variable by scanning the items (usually directories) withing the enviroment variable value. Each time an item is found that is a symbolic link, the item is replaced with its defererenced value. If no symbolic links are found, then the value of the environment variable is unchanged.

dump <envvar> Parses the environment variable value into items (usually directories) and outputs each item on a separate line.

erase <envvar> @<position-number> Modifies the environment variable by eliminating the item in the specifid position (starting with 1). If the position number is greater than the number of items, then the value of the environment variable is not modified.

erase <envvar> <item-or-dir> [<number-of-times>] Looks for the specified item (usually a directory) within the environment variable value and eliminates it up to the specified number of times (in case the item was listed more than once). The default number of times is 1. If the item is not found, then the value of the environment variable is not modified. (See also the "remove" command).

find <envvar> <item-or-dir> Looks for an item (usually a directrory) within the value of an environment variable and outputs the position number (starting with 1) at which the item was first found. If the item is not found, then nothing is output.

get <envvar> [@]<position-number> Outputs the position-number specified nth item (usually a directory) within the environment variable. The position number starts with 1 (one). If position number is greater than the count of items, then nothing is output.

insert <envvar> <new-item-or-dir> before|after <old-item-or-dir> Modifies the value of the environment variable such that the specified new item (usually a directory) is placed immediately before (if "before" was specified) or after (if "after is specified) the first location of the old item (also usually a diectory). It the old item is not found, then the new item is appended to the end.

insert <envvar> <new-item-or-dir> before|after @<position-number> Modifies the value of the environment variable such that the specified new item (usually a directory) is placed immediately before (if "before" was specified) or after (if "after is specified) the numeric position number specified. It the position number is greater than the number of items, then the new item is appended to the end.

prepend <envvar> <item-or-dir> Modifies the value of the environment variable so that the item (usually a directory) is placed as the very fist item of the new environment variable value.

remove <envvar> <item-or-dir> [<number-of-times>] Looks for the specified item (usually a directory) or any symbolic link equivalent within the environment variable value and eliminates it up to the specified number of times (in case the item or equivalent was listed more than once). The default number of times is 1. If the item is not found, then the value of the environment variable is not modified. (See also the "erase" command).

remove <envvar> @<position-number> Modifies the environment variable by eliminating the item in the specifid position (starting with 1). If the position number is greater than the number of items, then the value of the environment variable is not modified.

reverse <envvar> Modifies the value of the environment variable by reversing the order that items (usually directories) are listed.

show <envvar> Displays both the raw value of the environment variable and also breaks apart the items (usually directories) within the environment variable value into items which are displayed as a numbered list, one item per line.

weed <envvar> Modifies the value of the environment variable by scanning the items (usually directories) withing the enviroment variable value. Each time an item is found that is directory or file that actually exists, it is eliminated. If all items exist, then the value of the environment variable is unchanged.

auxenv exit status

auxenv completes with an exit status of 0 on success or with an exit status of 1 if an error was detected.

auxenv description

auxenv is a sourceable bash script which solves many problems working with path-like environment variables such as PATH, MANPATH, LD_LIBRARY_PATH, and so forth. It can display current settings in an easy to read form. It can add, delete, and reorder directory positions, it can weed out duplicate or nonexistent directories, and much more. The power of auxenv allows system administrators and end-users to create more robust and flexible login initialization scripts. Auxenv is also very useful to anyone who needs to switch back and forth between different versions of a programming language, enhancing or replacing the Linux "scl" utility.

auxenv is part of the auxilium installation package.

auxenv examples

Here are some examples of using auxenv.

- Displaying a pathlike environment variable:

  ```
  $ export DEMOPATH="/dir1:/dir2:/dir3"
  $ auxenv show DEMOPATH
  DEMOPATH is set to:
  /dir1:/dir2:/dir3
  Item breakdown:
    1=/dir1
    2=/dir2
    3=/dir3
  $ auxenv dump DEMOPATH
  /dir1
  /dir2
  /dir3
  ```

- Prepending a directory:

  ```
  $ export DEMOPATH="/dir1:/dir2:/dir3"
  $ auxenv prepend DEMOPATH /dir0
  $ echo "$DEMOPATH"
  /dir0:/dir1:/dir2:/dir3
  ```

- Inserting a directory:

  ```
  $ export DEMOPATH="/dir1:/dir2:/dir3"
  $ auxenv insert DEMOPATH "/dir1a" before "/dir2"
  $ echo "$DEMOPATH"
  /dir1:/dir1a:/dir2:/dir3
  ```

- Erasing a directory by name:

  ```
  $ export DEMOPATH="/dir1:/dir2:/dir3"
  $ auxenv erase DEMOPATH "/dir2"
  $ echo "$DEMOPATH"
  /dir1:/dir3
  ```

- Erasing a directory by position:

  ```
  $ export DEMOPATH="/dir1:/dir2:/dir3"
  $ auxenv erase DEMOPATH @2
  $ echo "$DEMOPATH"
  /dir1:/dir3
  ```

- Working with alternate separators:

  ```
  $ export DEMOPATH="red|green|blue"
  $ auxenv --ifs="|" show DEMOPATH
  DEMOPATH is set to:
  red|green|blue
  Item breakdown:
    1=red
    2=green
    3=blue
  ```

auxenv notes

Note that the auxenv "command" is not actually an "executable" script or binary image. Rather it is a "sourceable" script, meaning it is designed to run when invoked by the bash "source" built-in command. However, by using the bash built-in "alias" command, we can define an alias for auxenv such that it mimicks the usage behavior of an actual executable program. For example, consider the alias definition:

```
alias auxenv="source /usr/local/bin/auxenv"
```

Once this alias is defined, interactive users and other scripts can invoke (source) the auxenv script simply by specifying "auxenv" followed by any desired additional auxenv parameters. (Assuming auxenv was installed in /usr/local/bin.) If auxilium was installed system-wide by your system administrator it is quite likely this alias will already be defined for you (probably in the /etc/bashrc file). You can type "alias" to see a list of all currently defined aliases. If this alias is not defined, you can add the alias definition in your personal $(HOME)/.bashrc file. Note that your aliases should be defined in $(HOME)/.bashrc file, not $(HOME)/.bash_profile.

Sourceable scripts are needed if the current process environment is to be modified, rather than a subprocess environment that executable programs always run in. Since auxenv potentially needs to modify the current process environment, it needs to be a sourceable script.

auxwhere

auxwhere - find files via path-like env vars

auxwhere introduction

An executable script. Helps find files by filename when you are not sure of the directory by searching though a list in a path-like environment variable. The auxwhere utility is similar to, but also different than, other non-auxilium utilities such as *which* and *whereis*. The auxwhere utility is very flexible and can be used for many purposes, but it is geared towards making it easier to use sourceable (but non-executable) scripts in a manner closer to executable scripts.

auxwhere usage

```
auxwhere [<option>...] <filename>...
  -e <envvar>    | --env=<envvar>     # add env var to dir search list
  -t <.filetype> | --type=<.filetype> # add file type to permutation list
  -s <subdir>    | --subdir=<subdir>  # add sub-directory to permutations
  -x             | --exec             # only match executable files
  -a             | --all              # display all matches
  -h             | --help             # display help
  -v             | --version          # display version
```

auxwhere options

-e <envname> | –env=<envname> Specifies an environment variable containing a path-like list of directories through which auxwhere will search. If not specified, auxwhere uses the PATH environment variable by default. If this option is specified multiple times on the command line than multiple environment variable directory lists can be searched.

-t <.filetype> | –type=<.filetype> Specifies a file type suffix to be appended to a file name when searching. By specifying this option multiple times multiple file type permutations can be searched for. For example, "–type=.tar –type=.tar.gz" will search for both <filename>.tar and <filename>.tar.gz in each directory.

-s <dir> | –subdir=<dir> Specifies a sub-directory to search within each directory. By specifying this option multiple times multiple sub directories can be searched. For example, "–subdir=man1 –subdir=man1p –subdir=man1x –env=MANPATH" will specify a search though each of those sub-directories for each directory listed in MANPATH.

-x | –exec Specifies that only executable files will be matched. Executable files have the "x" bit set in their file protection mode.

-a | –all Specifies that all matches found should be listed, one per line. By default, only the first match is listed.

-w | –how Specifies that additional information about how a match was found should be indicated. This is usually an indication of which environment variable lead to the match. Each match listed (one or all) is output as the file specification of the matched file, followed by a tab character, followed by a short "hoe message" on the same line.

–ifs="<char>" Specifies the character which separates items within the environment variable value. For example, on Linux directories within the PATH ennvironment variable are separated by a colon (":") character. A colon (":") is the default separator if this option is not specified.

– Forces all following parameters to be considered filenames. This is only needed if you have are looking for a filename that migfht be confused with a command line option, for example, a file actually named "–help".

-h | –help Displays a brief auxwhere command usage summary. If used, this option should be the only option or parameter on the command line.

-v | –version Displays auxwhere version information. If used, this option should be the only option or parameter on the command line.

auxwhere exit status

auxwhere completes with an exit status of 0 if no errors are detected whether or not any matches were found. If some other error if found a non-zero exit status is returned.

auxwhere description

auxwhere is an executable bash script which can find a file by filename searching across directories in path-like environment variables. By default, the PATH variable is used, but other environment variables can also be specified. Permutations of sub-directories and file types can also be specified. By default, only the file specification of the first match is output, but if optionally requested, then all matches can be output, one file specification per line. If no matching files are found, then nothing is output.

auxwhere is part of the auxilium installation package.

auxwhere examples

Here are some examples of using auxwhere.

- Finding the first instance of a file somewhere in a PATH directory:

  ```
  $ auxwhere myfile
  /some/directory/myfile
  ```

- Finding all instances of a file somewhere in a PATH directory:

  ```
  $ auxwhere --all myfile
  /some/directory/myfile
  /some/other/directory/myfile
  /another/directory/myfile
  ```

- Finding the first instance of a file somewhere in a LD_LIBRARY_PATH directory.

  ```
  $ auxwhere --env=LD_LIBRARY_PATH libproxy.so.1
  /usr/lib64/libproxy.so.1
  ```

- Finding a file and seeing how the file was found using two path-like environment variables.

  ```
  $ auxwhere --all --env=PRIMARYPATH --env=SECONDARYPATH myfile.bash
  /some/directory/myfile.bash    (Found via PRIMARYPATH)
  /another/directory/myfile.bash     (Found via SECONDARYPATH)
  ```

- Finding a file searching multiple file types and a sub-directory.

  ```
  $ auxwhere --env=MANPATH --subdir=.man --type=.1 --type=.1.gz awk
  /usr/share/man/man1/awk.1.gz
  ```

auxsource

auxsource - enhanced bash source functionality

auxsource introduction

A sourceable script. Implements an extended version of the bash built-in *source* command. This also can make using sourceable (non-executable) scripts more like using executable scripts.

auxsource usage

```
auxsource [<option>...] [--] <file> [<arg>...]
  -h | --help           # display help
  -v | --version        # display version
```

(Assuming 'auxsource' is an alias to 'source <dir>/auxsource'.)

auxsource options

-h | –help Displays a brief auxenv command usage summary. If used, this option must be the only option or parameter on the command line.

-v | –version Displays auxenv version information. If used, this option must be the only option or parameter on the command line.

– Indicates that all remaining parameters are for the sourceable file. You will probably never need to use this, but if for some bizarre reason you had a file named "–help", for example, the only way to auxsource it would be to specify "– –help" instead of just "–help".

auxsource exit status

auxsource completes with an exit status of 0 if the final status of the file sourced was 0, otherwise auxsource completes with a non-zero exit status.

auxsource description

auxsource is a sourceable bash script which extends the capabilities of the built-in bash command "source". Its primary feature is that if a relative file specification is specified (a filename with no directory or a relative directory) then it will search through a path-list (using the PATH environment variable by default) and source the first match found. This makes working with sourceable scripts a little more like working with executable scripts or programs.

auxsource uses the auxwhere executable script (also part of the auxilium distribution) to find the exact file to be sourced. Command line parameters on the auxsource command line cannot be passed to auxwhere because they are reserved for use by whatever sourceable script is being invoked. However, the "AUXSOURCE_OPTIONS" environment variable may be set to whatever auxwhere options are desired. This allows full control over how sourceable files are located.

auxsource is part of the auxilium installation package.

auxsource examples

Here are some examples of using auxsource.

- Sourcing a file somewhere in a PATH directory:

  ```
  $ auxsource myfile
  ```

- Sourcing a file somewhere in a PATH directory and passing some parameters:

  ```
  $ auxsource myfile say "hello world"
  ```

- Sourcing a file specified by absolute location and passing parameters:

  ```
  $ auxsource /home/alice/mything.bash "more" "parameters"
  ```

- Sourcing a file somewhere in a MYPATH directory:

  ```
  $ export MYPATH="/home/alice:/home/bob"
  $ export AUXSOURCE_OPTIONS="--env=MYPATH"
  $ auxsource myfile "some" "more parameters"
  ```

auxsource notes

Note that the auxsource "command" is not actually an "executable" script or binary image. Rather it is a "sourceable" script, meaning it is designed to run when invoked by the bash "source" built-in command. However, by using the bash built-in "alias" command, we can define an alias for auxsource such that it mimicks the usage behavior of an actual executable program. For example, consider the alias definition:

```
alias auxsource="source /usr/local/bin/auxsource"
```

Once this alias is defined, interactive users and other scripts can invoke (source) the auxsource script simply by specifying "auxsource" followed by any desired additional auxsource parameters. (Assuming auxsource was installed in /usr/local/bin.) If auxilium was installed system-wide by your system administrator it is quite likely this alias will already be defined for you (probably in the /etc/bashrc file). You can type "alias" to see a list of all currently defined aliases. If this alias is not defined, you can add the alias definition in your personal $(HOME)/.bashrc file. Note that your aliases should be defined in $(HOME)/.bashrc file, not $(HOME)/.bash_profile.

Sourceable scripts are needed if the current process environment is to be modified, rather than a subprocess environment that executable programs always run in. Since auxsource potentially needs to modify the current process environment, it needs to be a sourceable script.

auxalias

auxalias - easy creation of auxilium aliases

auxalias introduction

A sourceable script. Facilitates creation of *aliases* for all auxilium sourceable scripts, which makes it possible to invoke a sourceable script like a command. It can create "static" aliases hard coded to a particular directory or "dynamic" aliases which dynamically use a path-like environment variable to find the right location of a sourcable script (much like how executable programs are located through PATH).

auxalias usage

```
source <dir>/auxalias [<option>...] # if not yet aliased
auxalias [<option>...]              # if already aliased
  -y | --dyn                        # create dynamic aliases
  -h | --help                       # display help
  -v | --version                    # display version
```

(Assuming 'auxalias' is an alias to 'source <dir>/auxalias'.)

auxalias options

-y | –dyn Define “dynamic aliases” rather than “static” aliases.

-h | –help Displays a brief auxalias command usage summary. If used, this option should be the only option or parameter on the command line.

-v | –version Displays auxalias version information. If used, this option should be the only option or parameter on the command line.

auxalias exit status

auxalias completes with an exit status of 0 on success or with an non-zero exit status if an error was detected.

auxalias description

auxalias is a sourceable bash script which defines either “static” or “dynamic” aliases for all auxilium sourceable scripts (auxenv, auxsource, and auxalias). A static alias is linked to a hardcoded directory. A dynamic alias uses a path-like environment variable (PATH by default) to dynamically determine each time it is invoked the first occurance of the target file to source. This mimicks the PATH behavior of executable scripts.

The auxalias script only defines aliases for auxilium sourcable scripts. The auxilium executable programs, such as auxwhere, auxchecktap, and auxguid do not need aliases.

auxalias is part of the auxilium installation package.

auxalias examples

Here are some examples of using auxalias.

- Setting up static auxilium aliases to installation in /usr/local:

  ```
  $ source /usr/local/bin/auxalias
  $ alias auxenv
  alias auxenv='source /usr/local/bin/auxenv'
  ```

- Setting up dynamic auxilium aliases which use PATH:

  ```
  $ source /usr/local/bin/auxalias
  $ alias auxenv
  alias auxenv='eval source $(auxwhere auxenv)'
  ```

auxalias notes

Note that the auxalias “command” is not actually an “executable” script or binary image. Rather it is a “sourceable” script, meaning it is designed to run when invoked by the bash “source” built-in command. However, by using the bash built-in “alias” command, we can define an alias for auxalias such that it mimicks the usage behavior of an actual executable program. For example, consider the alias definition:

```
alias auxalias="source /usr/local/bin/auxalias"
```

Once this alias is defined, interactive users and other scripts can invoke (source) the auxalias script simply by specifying “auxalias” followed by any desired additional auxalias parameters. (Assuming auxalias was installed in /usr/local/bin.) If auxilium was installed system-wide by your system administrator it is quite likely this alias will already be defined for you (probably in the /etc/bashrc file). You can type “alias” to see a list of all currently defined aliases. If this alias is not defined, you can add the alias definition in your personal $(HOME)/.bashrc file. Note that your aliases should be defined in $(HOME)/.bashrc file, not $(HOME)/.bash_profile.

Sourceable scripts are needed if the current process environment is to be modified, rather than a subprocess environment that executable programs always run in. Since auxalias potentially needs to modify the current process environment, it needs to be a sourceable script.

auxchecktap

auxchecktap - check and summarize TAP unit test results

auxchecktap introduction

An executable script. Facilitates integration and checking of unit test results. Most unit test tools are targeted towards a particular programming language, which means most organizations have multiple unit test tools. Fortunately, there is a widely adopted format standard for unit test result file called "TAP", which stands for "Test Anything Protocol" (see https://testanything.org/). Most modern unit test tools can product TAP format unit test result files. The auxchecktap utility scans these TAP format unit test files and can optionally create a summary TAP unit test results file. This makes it possible to create tiered hierarchically unit test summary reports. This utility also checks unit test results files much more consistently and carefully than is reasonably human possible. Error lines buried in the middle of long files are always detected, abmormal early test suite termination without error messages are detected, test sequence problems are detected, and so forth.

auxchecktap usage

```
auxchecktap [<option>...] [<filename>...]
  -s | --strict   # strict TAP parsing (default)
  -n | --nostrict # less strict TAP parsing
  -b | --brief    # do not output # error comment lines
  -d | --debug    # output debug info to stderr
  -h | --help     # display help
  -v | --version  # display version
```

auxchecktap options

-s | –strict Specifies that strict TAP format parsing rules will be followed. Poorly formatted unit test results will result in an error being reported. This is the default.

-n | –nostrict Specifies that less restrictive TAP format parsing rules will be used. This sometimes helps when parsing rersult from unit test tools that only loosely follow TAP formatting rules.

-b | –brief Specifies that when producing summary TAP results, a "not ok" result line will be produce when a problem is detected, but the optional "#" comment lines following the report line which describe the problem will not be produced.

-d | –debug Specifies that extra debug information should be outrput to standard error.

-h | –help Displays a brief auxchecktap usage summary. If used, this option should be the only option or parameter on the command line.

-v | –version Displays auxchecktap version information. If used, this option should be the only option or parameter on the command line.

auxchecktap exit status

auxchecktap completes with an exit status of 0 if no errors are detected and no failed of missing unit test results were uncovered. It completes with a non-zero status is there were any usage errors, or if any fail uni tests were reported, or if missing unit tests were uncovered.

auxchecktap description

auxchecktap is an executable bash script which scans TAP format unit test results read from either standard input or one or more files. It outputs a summary of the scan to standard output also in TAP format. The summary includes one “TAPFILE” report line per log file scanned. The summary also includes one “TAPTOTAL” summary line with totals for all the log files auxtapcheck just finished scanning. Finally, the summary also includes one “TAPGRAND” summary line with grand totals for the just scanned files plus any indirect “TAPGRAND” totals that the log files might have had embedded in them. Thus, if auxchecktap scans the output of other auxchecktap logs the TAPGRAND summary line indicates the total picture. This allows a hierarchy of unit test results to be quickly and easily checked.

auxchecktap is part of the auxilium installation package.

auxchecktap examples

Here are some examples of using auxchecktap.

- Scanning a unit test results log file:

  ```
  $ auxchecktap example1.taplog
  1..3
  ok 1 [TAPFILE: '/home/tester/test/example1.taplog' 89 tests, 0 errors]
  ok 2 [TAPTOTAL: 1 file, 89 tests, 0 errors]
  ok 3 [TAPGRAND: 1 file, 89 tests, 0 errors]
  1..1
  ok 1 /home/tester/test/auxilium.taplog: 85 tests.
  $ echo "$?"
  0
  ```

- Scanning multiple unit test result log files:

  ```
  $ auxchecktap example*.taplog
  1..6
  ok 1 [TAPFILE: '/home/tester/test/example1.taplog' 89 tests, 0 errors]
  ok 2 [TAPFILE: '/home/tester/test/example2.taplog' 88 tests, 0 errors]
  ok 3 [TAPFILE: '/home/tester/test/example3.taplog' 40 tests, 0 errors]
  ok 4 [TAPFILE: '/home/tester/test/example4.taplog' 20 tests, 0 errors]
  ok 5 [TAPTOTAL: 4 files, 237 tests, 0 errors]
  ok 6 [TAPGRAND: 4 files, 237 tests, 0 errors]
  $ echo "$?"
  0
  ```

- Scanning a unit test log file with a failed unit test report in it:

  ```
  $ auxchecktap another.taplog
  1..3
  not ok 1 [TAPFILE: '/home/tester/test/another.taplog' 3 tests, 1 error]
  # Found 1 failed test.
  not ok 2 [TAPTOTAL: 1 file, 3 tests, 1 error]
  not ok 3 [TAPGRAND: 1 file, 3 tests, 1 error]
  ```

```
$ echo "$?"
1
```

auxguid

auxguid - generate a 32 hex character (128 bit) GUID/UUID

auxguid introduction

An executable script. This script generates a pseudo random 128-bit (32 hex character) GUID (Globally Unique IDentifier) sometimes also known as a UUID (Universally Unique IDentifier). There are some standards for GUID/UUID generation, but none are universal and even the definition of GUID and UUID is under debate (see https://en.wikipedia.org/wiki/Universally_unique_identifier). This script uses its own algorithm to generate the GUID. The GUID is not cryptogrphically random, but it is sufficiently random such that the chances of any two independently generated GUIDs ever being the same (colliding) is astronomically low. Most programming languages have GUID/UUID generation facilities that are probably more standardized, but it is sometimes really convenient to have a good GUID generater that does not need anything but bash.

auxguid usage

```
auxguid [<option>...]
  -l | --lower    # use lowercase hex letters
  -u | --upper    # use uppercase hex letters
  -w | --warn     # display non-fatal warnings to stderr
  -d | --debug    # output debug info to stderr
  --uuidgen       # use the uuidgen utility if installed
  -h | --help     # display help
  -v | --version  # display version
```

auxguid options

-l | –lower Specifies that lowercase hex characters will be used in the generated GUID/UUID.

-u | –upper Specifies that uppercase hex characters will be used in the generated GUID/UUID.

-w | –warn Specifies that warning messages about non-fatal conditions should be output to standard error.

-d | –debug Specifies that extra debug information should be outrput to standard error.

–**uuidgen** Specifies that the optional Linux program “uuidgen” should be used if it is installed. If uuidgen is not install, auxguid will still generated a GUID/UUID with its internal algorithm.

-h | –help Displays a brief auxguid usage summary. If used, this option should be the only option or parameter on the command line.

-v | –version Displays auxguid version information. If used, this option should be the only option or parameter on the command line.

auxguid exit status

auxguid completes with an exit status of 0 if it successfully generates and outputs a GUID/UUID, otherwise a non-zero exit status is returned.

auxguid description

auxguid is an executable bash script which, by default, uses a custom algorithm to generate a 32 hex xharacter 128 bit GUID (Globally Unigue IDentifier) sometimes also called a UUID (Universally Unique IDentifier) using nothing by bash. (See https://en.wikipedia.org/wiki/Universally_unique_identifier).

Optionally, if the "–uuidgen" option is specified and if the optional Linux program "uuidgen" is installed then that program is used to generate the GUID/UUID.

auxguid is part of the auxilium installation package.

auxguid examples

Here are some examples of using auxguid.

- Generate a GUID/UUID:

  ```
  $ auxguid
  2c7459630bbf26bc01ee5a33259a215a
  ```

- Generate a lowercase GUID/UUID:

  ```
  $ auxguid --lower
  5bb659630c1f28006a2d5a57259a215a
  ```

- Generate an uppercase GUID/UUID:

  ```
  $ auxguid --upper
  39EB59630C67184D7E0E5B1E259A215A
  ```

Platforms

Auxilium programs are all written in Bash, the standard Linux shell language. Bash is an interpretive language, so as long as your platform has a relatively new version of Bash installed, it *should* be able to run the runtime Auxilium Bash scripts. There is no need of different runtime versions of Auxilium for different hardware platforms because there are no hard-compiled programs.

That being said, to date Auxilium has only been tested on Linux platforms. The author, however, would be very interested in user experience reports.

Authors

Eric Alan Christiansen
eric@motleybytes.com
https://www.linkedin.com/in/ericchristiansen/

Legal

Copyright

Licensing

See Also

- https://github.com/ericmotleybytes/auxilium : The master auxilium GitHub repository. You can also report bugs, describe issues, and request enhancements here.
- https://testanything.org/ : A description of TAP, the unit testing Test Anything Protocol.