Smoke testing of all base utilities

Automating generation of test scripts

Overview

I have been brainstorming since some time on how to proceed to write an automation tool which is supposed to produce **smoke tests** with a **minimal** set of test-cases for all the utilities in the base system. Hoping to get valuable suggestions before proceeding on implementation.

Procedure

The previously written functional test for `ls(1)` at the time of project proposal submission (the proposal was written in 4 days, hence might be error-prone) was the first step to demonstrate a very simple test which checks for basic options supported by a utility and executes them, reporting success or failure.

A successful execution of a command of the form -

```
>> utility -<short_option> --<long_option>
```

will denote that the utility under test is properly linked.

We define a set containing options which can be easily tested. An example set can contain the following options -

```
['--version', '--help',.., '-v', '-h']
```

Note 1: `'-h'` also refers to *human readable format* for some utilities, hence should be taken into account.

Note 2: Some utilities like `dd` don't accept any arguments, hence an approach for covering them has to be figured out.

The set can be broken down into two separate sets containing short-options and long-options to effectively reduce the search time while performing the next step, so that lookups for short and long options can be done separately.

For each utility:

> Pass an unsupported option (can be chosen experimentally) which "might" produce a usage message that can be parsed.

If the above step fails, then:

> Parse the relevant man-page(s) for the supported options.

On finding options which are defined in the set, we include them in the utility's `short_options[]` / `long_options[]` array.

Complexity analysis

A rough complexity analysis for the running time of the tool (there may be a chance of improvement) -

- n - total number of utilities
  (**~ 567** remaining without test coverage)
- d - total number of (short & long) options in the set
- m - "upper bound" on the number of options a utility may support
- l - maximum length of any supported long option

Total running time complexity = $O(n * d * \underbrace{m * l}) $

It is currently assumed that string matching (for long options) will be done in $O(l)$ time (list-like behavior). However, $O(m * l)$ will be reduced to $O(1)$ lookup time when using for e.g. a set in python.

Reporting failures

The tool will report the commands which failed, hence denoting that the utility under test is not properly linked.

Note: A point to be noted here is that following the above mentioned plan, the ability of our tool to figure out the **reasons of failure** will be out of scope i.e. regression testing of a utility will not be possible.

Reference

> *On Thu, Jun 8, 2017 at 11:46 PM Brooks Davis <brooks@freebsd.org> wrote:*
> The intent of the project really was the former, hence the focus on
> some automation in the project idea. I think what you've been doing is
> also useful and doing a bit more of it will help prepare you for the
> automated part.
>
> *On Wed, Jun 07, 2017 at 07:47:42PM +0000, Shivansh Rai wrote:*
> > Hello all,
> >
> > I have one query concerning the overall aim of the project - is it supposed
> > to provide a "minimal" test coverage to "all" the base utilities or a
> > rigorous test coverage covering most of the supported options.
> > If we choose the former, the rate of increment of the "smoke tests score"
> > will be high, and relatively slower for the latter case (the chances for us
> > to cover all of them until final evaluations are relatively less in this
> > case since around 568 are left).