KSConf Documentation

Release 0.6.2

Lowell Alleman

Feb 09, 2019

Contents

Author Lowell Alleman (Kintyre)

Version 0.6

CHAPTER 1

Welcome to KSCONF!

KSCONF in a modular command line tool for Splunk admins and app developers. It's quick and easy to get started with basic commands and grow into the more advanced commands as needed. Check out our growing body of documentation to help smooth your transition into a more-manged Splunk environment, or explore ways to integrate ksconf's capabilities into your existing workflow.

No matter where you're starting from, we think ksconf can help! We're glad your here. Let us know if there's anything we can do to help along your journey.

– Kintyre team

CHAPTER 2

Install

Ksconf can be directly installed as a Python (via `pip`) or as a Splunk app. The Splunk app option is often easier.

To install as a **python package**, run the following:

```
pip install kintyre-splunk-conf
```

To install the **Splunk app**, download the latest KSCONF App for Splunk release. Note that a one-time registration command is need to make `ksconf` executable:

```
splunk cmd python $SPLUNK_HOME/etc/apps/ksconf/bin/bootstrap_bin.py
```

CHAPTER 3

User Guide

3.1 Introduction

KSCONF (Kintyre's Splunk Configuration tool) is a command-line tool that helps administrators and developers manage their Splunk environments by enhancing their ability to control configuration files. By design, the interface is modular so that each function (aka subcommand) can be learned quickly and used independently. Most Ksconf commands are simple enough for a quick one-off job, yet reliable enough to integrate into complex app build and deployment workflow.

Ksconf helps manage the nuances with storing Splunk apps in a version control system, like git. It also supports pointing live Splunk apps to a working tree, merging changes from the live system's (local) folder to the version controlled folder (often 'default'), and in more complex cases, it deals with more than one *layer* of "default", which Splunk can't handle natively).

Note: What KSCONF is not

Ksconf does *not* replace your existing Splunk deployment mechanisms or version control tools. The goal is to complement and extend, not replace, the workflow that work for you.

3.1.1 Design principles

Ksconf is a toolbox. Each tool has a specific purpose and function that works independently. Borrowing from the Unix philosophy, each command should do one thing well and be easily combined to handle higher-order tasks.

When possible, be familiar. Various commands borrow from popular UNIX command line tools such as `grep` and `diff`. The modular nature of the command and other design features were borrowed from `git` and `splunk` as well.

Don't impose workflow. Ksconf works with or without version control and independently of your deployment mechanisms. If you are looking to implement these things, ksconf is a great building block.

Embrace automated testing. It's impractical to check every scenarios between each release, but significant work has gone into unittesting the CLI to avoid breaks between releases.

3.1.2 Common uses for ksconf

- Promote changes from `local` to `default`
- Maintain multiple independent layers of configurations
- Reduce duplicate settings in a local file
- Upgrade apps stored in version control
- Merge or separate configuration files
- Git pre-commit hook for validation
- Git post-checkout hook for workflow automation
- Send *.conf* stanzas to a REST endpoint (Splunk Cloud or no file system access)

3.2 Concepts

3.2.1 Configuration layers

The idea of configuration layers are used is shared across multiple actions in ksconf. Specifically, *combine* is used to merge multiple layers, and the *unarchive* command can be used to install or upgrade an app in a layer-aware way.

What's the problem?

In a typical enterprise deployment of Splunk, a single app can easily have multiple logical sources of configuration:

1. Upstream app developer (typically via Splunkbase)
2. Local developer app-developer adds organization-specific customizations or fixes
3. Splunk admin tweaks the inappropriate `indexes.conf` settings, and
4. Custom knowledge objects added by your subject matter experts.

Ideally we'd like to version control these, but doing so is complicated because normally you have to manage all 4 of these logical layers in one 'default' folder.

Note: Isn't that what the **local** folder is for?

Splunk requires that app settings be located either in 'default' or 'local'; and managing local files with version control leads to merge conflicts; so effectively, all version controlled settings need to be in 'default', or risk merge conflicts.

Let's suppose a new upstream version is released. If you aren't managing layers independently, then you have to manually upgrade the app being careful to preserve all custom configurations. Compare this to the solution provided by the 'combine' functionality. Because logical sources can be stored separately in their own directories changes can managed independently. The changes in the "upstream" layer will only ever be from official release; there's no combing through the commit log to see what default was changed to figure out what custom changes need to be preserved and reapplied.

While this doesn't completely remove the need for a human to review app upgrades, it does lower the overhead enough that updates can be pulled in more frequently, thus reducing the divergence potential. (Merge frequently.)

3.2.2 Minimizing files

A typical scenario & why does this matter:

To customizing a Splunk app or add-on, many admins simply start by copying the conf file from default to local and then applying your changes to the local file. That's fine, but if you stopping here you mave have just complicated future upgrades. This is because the local file doesn't contain *just* your settings, it contains all the default settings too. So in the futre, fixes published by the app creator may be masked by your local settings. A better approach is to reduce the local conf file leaving only the stanzas and settings that you indented to change. This make your conf files easier to read and makes upgrades easier, but it's tedious to do by hand. Therefore, take a look at the exact problem that the *minimize* command addresses.

Important: *Why all the fuss?* From the splunk docs

> "When you first create this new version of the file, **start with an empty file and add only the attributes that you need to change.** Do not start from a copy of the default directory. If you copy the entire default file to a location with higher precedence, any changes to the default values that occur through future Splunk Enterprise upgrades cannot take effect, because the values in the copied file will override the updated values in the default file." – *[SPLKDOC1]*.

3.3 Installation Guide

KSCONF can be installed either as a Splunk app or a Python package. Picking the option that's right for you is typically fairly easy.

Unless you have experence with Python packaging or are planning on customizing or extending ksconf then *Splunk app* is likely the best place for you to start. If you go with the native Python option, then you have many additional decisions to make.

Note: The introduction of a Splunk app is a fairly new situation (as of the 0.6.x release.) Originally we resisted this idea, since `ksconf` was designed to manage other apps, not live within one. But ultimately, the packaging decision was driven by the bombardment of complexity encountered with nearly every install. Python packaging is a mess and daunting for the uninitiated.

3.3.1 Overview

Install	Advantages	Potential pitfalls
Python package	• Most 'pure' and flexible install • One command install. (ideal) • Easy upgrades • More extendable (plugins) • *Install Python package*	• Lots of potential variations and pitfalls • Many Linux distro's don't ship with `pip` • Must consider/coordinate installation user. • Often requires some admin access. • Too many install options (complexity)
Splunk app	• Quick installation (single download) • Requires one time bootstrap command • Self contained; no admin access require • Fast demo; fight with `pip` later • *Install Splunk App*	• Crippled Python install (no `pip`) • Can't add custom extensions (entrypoints) • No CLI completion (yet) • *Grandfather Paradox*
Offline package	• Security: strict review and change control • *Advanced Installation Guide*.	• Requires many steps. • Inherits 'Python package' pitfalls.

3.3.2 Requirements

Python package install:

- Python Supports Python 2.7, 3.4+
- PIP (strongly recommended)
- Tested on Mac, Linux, and Windows

Splunk app install:

- Splunk 6.0 or greater is installed

3.3.3 Install Splunk App

Download and install the KSCONF App for Splunk. Then open a shell, switch to the Splunk user account and run this one-time bootstrap command.

```
splunk cmd python $SPLUNK_HOME/etc/apps/ksconf/bin/bootstrap_bin.py
```

This will add `ksconf` to Splunk's `bin` folder, thus making it executable either as `ksconf` or worse case `splunk cmd ksconf`. (If you can run `splunk` without giving it a path, then `ksconf` should work too.)

At some point we may add an option for you to do this setup step from the UI.

Note: Alternate download

You can also download the latest (and pre-release) SPL from the GitHub Releases page. Download the file named like `ksconf-app_for_splunk-ver.tgz`

3.3.4 Install Python package

Quick install

Using pip:

```
pip install kintyre-splunk-conf
```

System-level install: (For Mac/Linux)

```
curl https://bootstrap.pypa.io/get-pip.py | sudo python - kintyre-splunk-conf
```

Enable Bash completion

If you're on a Mac or Linux, and would like to enable bash completion, run these commands:

```
pip install argcomplete
echo 'eval "$(register-python-argcomplete ksconf)"' >> ~/.bashrc
```

(Currently for Splunk APP installs; not because it can't work, but because it's not documented or tested yet. Pull request welcome.)

Ran into issues?

If you run into any issues, then please dive into the *Advanced Installation Guide*. Much time and effort was placed into compiling that information from all the scenarios we encounted, so please check it out. You may want to start under the *Troubleshooting*.

3.3.5 Install from GIT

If you'd like to contribute to ksconf, or just build the latest and greatest, then install from the git repository is a good choice. (Technically this is still installing with `pip`, so it's easy to switch between a PyPI install, and a local install.)

```
git clone https://github.com/Kintyre/ksconf.git
cd ksconf
pip install .
```

See *Developer setup* for additional details about contributing to ksconf.

3.3.6 Validate the install

No matter how you install `ksconf`, you can confirm that it's working with the following command:

```
ksconf --version
```

The output should look something like this:

```
                                  #
                                  ##
 ###  ##      #### ###### #######  ###  ##  #######
 ### ##     ###  ###           ## #### ##
 #####      ###  ###      ##   ## #######  #######
 ### ##     ###  ###      ##   ## ### ###  ##
 ###  ## #####    ######   #####  ###  ##  ##
                                        #

ksconf 0.6.1  (Build 252)
Python: 2.7.15  (/Applications/splunk/bin/python)
Git SHA1 dd218785 committed on 2019-02-07
Written by Lowell Alleman <lowell@kintyre.co>.
Copyright (c) 2019 Kintyre Solutions, Inc.
Licensed under Apache Public License v2

Commands:
    check            (stable, from Distribution('kintyre_splunk_conf', '0.6.1-py2.7'))
    combine          (beta, from Distribution('kintyre_splunk_conf', '0.6.1-py2.7'))
    diff             (stable, from Distribution('kintyre_splunk_conf', '0.6.1-py2.7'))
    filter           (alpha, from Distribution('kintyre_splunk_conf', '0.6.1-py2.7'))
    merge            (stable, from Distribution('kintyre_splunk_conf', '0.6.1-py2.7'))
    minimize         (beta, from Distribution('kintyre_splunk_conf', '0.6.1-py2.7'))
```

(continues on next page)

(continued from previous page)

```
    promote          (beta, from Distribution('kintyre_splunk_conf', '0.6.1-py2.7'))
    rest-export      (beta, from Distribution('kintyre_splunk_conf', '0.6.1-py2.7'))
    snapshot         (alpha, from Distribution('kintyre_splunk_conf', '0.6.1-py2.7'))
    sort             (stable, from Distribution('kintyre_splunk_conf', '0.6.1-py2.7'))
    unarchive        (beta, from Distribution('kintyre_splunk_conf', '0.6.1-py2.7'))
```

If you run into any issues, check out the *Validate the install*

3.3.7 Command line completion

Bash completion allows for a more intuitive interactive workflow by providing quick access to command line options and file completions. Often this saves time since the user can avoid mistyping file names or be reminded of which command line actions and arguments are available without switching contexts. For example, if the user types `ksconf d` and hits `Tab` then the `ksconf diff` is completed. Or if the user types `ksconf` and hits `Tab` twice, the full list of command actions are listed.

This feature uses the argcomplete Python package and supports Bash, zsh, tcsh.

Install via pip:

```
pip install argcomplete
```

Enable command line completion for ksconf can be done in two ways. The easiest option is to enable it for ksconf only. (However, it only works for the current user, it can break if the ksconf command is referenced in a non-standard way.) The alternate option is to enable global command line completion for all python scripts at once, which is preferable if you use this module with many python tool.

Enable argcomplete for ksconf only:

```
# Edit your bashrc script
vim ~.bashrc

# Add the following line
eval "$(register-python-argcomplete ksconf)"

# Restart you shell, or just reload by running
source ~/.bashrc
```

To enable argcomplete globally, run the command:

```
activate-global-python-argcomplete
```

This adds new script to your the `bash_completion.d` folder, which can be use for all scripts and all users, but it does add some minor overhead to each completion command request.

OS-specific notes:

- **Mac OS X**: The global registration option has issue due the old version of Bash shipped by default. So either use the one-shot registration or install a later version of bash with homebrew: `brew install bash` then. Switch to the newer bash by default with `chsh /usr/local/bin/bash`.

- **Windows**: Argcomplete doesn't work on windows Bash for GIT. See argcomplete issue 142 for more info. If you really want this, use Linux subsystem for Windows instead.

3.4 Commands

The ksconf command documentation is provided in the following ways:

1. A detailed listing of each sub-command is provided in this section. This includes relevant background descriptions, typical use cases, examples, and discussion of relevant topics. An expanded descriptions of CLI arguments and their usage is provided here. If you've not used a particual command before, start here.
2. The *Command line reference* provides a quick an convenient reference when the command line is unavailable. The same information is available by typing `ksconf --help`. This is most helpful if you're already familiar with a command, but need a quick refresher.

Warning: Apologies for the dust

The command docs are currently undergoing reorganization. We're considering a topical layout rather than a per-command layout. Feedback and technical writing / organization contributions are highly welcomed.

Table 1: Command Listing

Command	Maturity	Description
ksconf check	stable	Perform basic syntax and sanity checks on .conf files
ksconf combine	beta	Combine configuration files across multiple source directories into a single destination directory. This allows for an arbitrary number of splunk configuration layers to coexist within a single app. Useful in both ongoing merge and one-time ad-hoc use.
ksconf diff	stable	Compare settings differences between two .conf files ignoring spacing and sort order
ksconf filter	alpha	A stanza-aware GREP tool for conf files
ksconf merge	stable	Merge two or more .conf files
ksconf minimize	beta	Minimize the target file by removing entries duplicated in the default conf(s)
ksconf promote	beta	Promote .conf settings from one file into another either in batch mode (all changes) or interactively allowing the user to pick which stanzas and keys to integrate. Changes made via the UI (stored in the local folder) can be promoted (moved) to a version-controlled directory.
ksconf rest-export	beta	Export .conf settings as a curl script to apply to a Splunk instance later (via REST)
ksconf snapshot	alpha	Snapshot .conf file directories into a JSON dump format
ksconf sort	stable	Sort a Splunk .conf file creating a normalized format appropriate for version control
ksconf unarchive	beta	Install or upgrade an existing app in a git-friendly and safe way

3.4.1 ksconf

Ksconf: Kintyre Splunk CONFig tool

This utility handles a number of common Splunk app maintenance tasks in a small and easy to deploy package. Specifically, this tools deals with many of the nuances with storing Splunk apps in git, and pointing live Splunk apps to a git repository. Merging changes from the live system's (local) folder to the version controlled (default) folder, and dealing with more than one layer of "default" (which splunk can't handle natively) are all supported tasks.

```
usage: ksconf [-h] [--version] [--force-color]
              {check,combine,diff,filter,merge,minimize,promote,rest-export,snapshot,sort,
↪unarchive}
              ...
```

Named Arguments

--version	show program's version number and exit
--force-color	Force TTY color mode on. Useful if piping the output a color-aware pager, like 'less -R'

3.4.2 ksconf check

Provide basic syntax and sanity checking for Splunk's .conf files. Use Splunk's builtin 'btool check' for a more robust validation of keys and values.

Consider using this utility as part of a pre-commit hook.

```
usage: ksconf check [-h] [--quiet] FILE [FILE ...]
```

Positional Arguments

FILE	One or more configuration files to check. If '-' is given, then read a list of files to validate from standard input

Named Arguments

--quiet, -q	Reduce the volume of output.

Note: Key concepts

Before diving into the `combine` command, it may be helpful to brush up on the concept of *configuration layers*.

3.4.3 ksconf combine

Merge .conf settings from multiple source directories into a combined target directory. Configuration files can be stored in a `/etc/*.d` like directory structure and consolidated back into a single 'default' directory.

This command supports both one-time operations and recurring merge jobs. For example, this command can be used to combine all users knowledge objects (stored in 'etc/users') after a server migration, or to merge a single user's settings after an their account has been renamed. Recurring operations assume some type of external scheduler is being used. A best-effort is made to only write to target files as needed.

The 'combine' command takes your logical layers of configs (upstream, corporate, splunk admin fixes, and power user knowledge objects, ...) expressed as individual folders and merges them all back into the single `default` folder that Splunk reads from. One way to keep the 'default' folder up-to-date is using client-side git hooks.

No directory layout is mandatory, but but one simple approach is to model your layers using a prioritized 'default.d' directory structure. (This idea is borrowed from the Unix System V concept where many services natively read their config files from `/etc/*.d` directories.)

```
usage: ksconf combine [-h] [--target TARGET] [--dry-run] [--banner BANNER]
                      source [source ...]
```

Positional Arguments

source	The source directory where configuration files will be merged from. When multiple sources directories are provided, start with the most general and end with the specific; later sources will override values from the earlier ones. Supports wildcards so a typical Unix `conf.d/##-NAME` directory structure works well.

Named Arguments

--target, -t	Directory where the merged files will be stored. Typically either 'default' or 'local'
--dry-run, -D	Enable dry-run mode. Instead of writing to TARGET, preview changes as a 'diff'. If TARGET doesn't exist, then show the merged file.
--banner, -b	A warning banner to discourage manual editing of conf files.

You may have noticed similarities between the `combine` and *merge* subcommands. That's because under the covers they are using much of the same code. The combine operations essentially does a recursive merge between a set of directories. One big difference is that `combine` command will gracefully handle non-conf files intelligently, not just conf files.

Note: Mixing layers

Just like all layers can be managed independently, they can also be combined in any way you'd like. While this workflow is out side the scope of the examples provided here, it's very doable. This also allows for different layers to be mixed-and-matched by selectively including which layers to combine.

Examples

Merging a multilayer app

Let's assume you have a directory structure that looks like the following. This example features the Cisco Security Suite.

```
Splunk_CiscoSecuritySuite/
├── README
├── default.d
│   ├── 10-upstream
│   │   ├── app.conf
│   │   ├── data
│   │   │   └── ui
│   │   │       ├── nav
│   │   │       │   └── default.xml
```

(continues on next page)

(continued from previous page)

```
│   │   │       └── views
│   │   │           ├── authentication_metrics.xml
│   │   │           ├── cisco_security_overview.xml
│   │   │           ├── getting_started.xml
│   │   │           ├── search_ip_profile.xml
│   │   │           ├── upgrading.xml
│   │   │           └── user_tracking.xml
│   │   ├── eventtypes.conf
│   │   ├── macros.conf
│   │   ├── savedsearches.conf
│   │   └── transforms.conf
│   ├── 20-my-org
│   │   └── savedsearches.conf
│   ├── 50-splunk-admin
│   │   ├── indexes.conf
│   │   ├── macros.conf
│   │   └── transforms.conf
│   └── 70-firewall-admins
│       ├── data
│       │   └── ui
│       │       └── views
│       │           ├── attacks_noc_bigscreen.xml
│       │           ├── device_health.xml
│       │           └── user_tracking.xml
│       └── eventtypes.conf
```

In this structure, you can see several layers of configurations at play:

1. The `10-upstream` layer appears to be the version of the default folder that shipped with the Cisco app.
2. The `20-my-org` layer is small and only contains tweaks to a few savedsearch entires.
3. The `50-splunk-admin` layer represents local settings changes to specify index configurations, and to augment the macros and transformations that ship with the default app.
4. And finally, `70-firewall-admins` contains some additional view (2 new, and 1 existing). Note that since `user_tracking.xml` is not a `.conf` file it will fully replace the upstream default version (that is, the file in `10-upstream`)

Here's are the commands that could be used to generate a new (merged) `default` folder from all these layers shown above.

```
cd Splunk_CiscoSecuritySuite
ksconf combine default.d/* --target=default
```

See also:

The *unarchive* command can be used to install or upgrade apps stored in a version controlled system in a layer-aware manor.

Consolidating 'users' directories

`combine` can consolidate 'users' directory across several instances after a phased server migration.

3.4.4 ksconf diff

Summary

Compare settings differences between two .conf files ignoring spacing and sort order

Compares the content differences of two .conf files

This command ignores textual differences (like order, spacing, and comments) and focuses strictly on comparing stanzas, keys, and values. Note that spaces within any given value will be compared. Multiline fields are compared in are compared in a more traditional 'diff' output so that long savedsearches and macros can be compared more easily.

```
usage: ksconf diff [-h] [-o FILE] [--comments] CONF1 CONF2
```

Positional Arguments

CONF1	Left side of the comparison
CONF2	Right side of the comparison

Named Arguments

-o, --output	File where difference is stored. Defaults to standard out.
--comments, -C	Enable comparison of comments. (Unlikely to work consistently)

Example

Add screenshot here

To use `ksconf diff` as an external diff tool, check out *Ksconf as external difftool*.

3.4.5 ksconf filter

Filter the contents of a conf file in various ways. Stanzas can be included or excluded based on provided filter, based on the presents or value of a key.

Where possible, this command supports GREP-like arguments to bring a familiar feel.

```
usage: ksconf filter [-h] [-o FILE] [--comments] [--verbose]
                     [--match {regex,wildcard,string}] [--ignore-case]
                     [--invert-match] [--files-with-matches]
                     [--count | --brief] [--stanza PATTERN]
                     [--attr-present ATTR] [--keep-attrs WC-ATTR]
                     [--reject-attrs WC-ATTR]
                     CONF [CONF ...]
```

Positional Arguments

CONF	Input conf file

Named Arguments

-o, --output	File where the filtered results are written. Defaults to standard out.
--comments, -C	Preserve comments. Comments are discarded by default.
--verbose	Enable additional output.
--match, -m	Possible choices: regex, wildcard, string Specify pattern matching mode. Defaults to 'wildcard' allowing for `*` and `?` matching. Use 'regex' for more power but watch out for shell escaping. Use 'string' enable literal matching.
--ignore-case, -i	Ignore case when comparing or matching strings. By default matches are case-sensitive.
--invert-match, -v	Invert match results. This can be used to show what content does NOT match, or make a backup copy of excluded content.

Output mode

Select an alternate output mode. If any of the following options are used, the stanza output is not shown.

--files-with-matches, -l	List files that match the given search criteria
--count, -c	Count matching stanzas
--brief, -b	List name of matching stanzas

Stanza selection

Include or exclude entire stanzas using these filter options.

All filter options can be provided multiple times. If you have a long list of filters, they can be saved in a file and referenced using the special `file://` prefix.

--stanza	Match any stanza who's name matches the given pattern. PATTERN supports bulk patterns via the `file://` prefix.
--attr-present	Match any stanza that includes the ATTR attribute. ATTR supports bulk attribute patterns via the `file://` prefix.

Attribute selection

Include or exclude attributes passed through. By default all attributes are preserved. Whitelist (keep) operations are preformed before blacklist (reject) operations.

--keep-attrs	Select which attribute(s) will be preserved. This space separated list of attributes indicates what to preserve. Supports wildcards.
--reject-attrs	Select which attribute(s) will be discarded. This space separated list of attributes indicates what to discard. Supports wildcards.

3.4.6 ksconf merge

Merge two or more .conf files into a single combined .conf file. This could be used to merge the props.conf file from ALL technology addons into a single file:

ksconf merge –target=all-ta-props.conf etc/apps/*TA*/{default,local}/props.conf

```
usage: ksconf merge [-h] [--target FILE] [--dry-run] [--banner BANNER]
                    FILE [FILE ...]
```

Positional Arguments

FILE	The source configuration file to pull changes from.

Named Arguments

--target, -t	Save the merged configuration files to this target file. If not provided. the the merged conf is written to standard output.
--dry-run, -D	Enable dry-run mode. Instead of writing to TARGET, preview changes in 'diff' format. If TARGET doesn't exist, then show the merged file.
--banner, -b	A banner or warning comment added to the top of the TARGET file. This is often used to warn Splunk admins from editing an auto-generated file.

3.4.7 ksconf minimize

See also:

See the *Minimizing files* for background on why this is important.

Minimize a conf file by removing the default settings

Reduce local conf file to only your indented changes without manually tracking which entries you've edited. Minimizing local conf files makes your local customizations easier to read and often results in cleaner add-on upgrades.

```
usage: ksconf minimize [-h] [--target TARGET] [--dry-run | --output OUTPUT]
                       [--explode-default] [-k PRESERVE_KEY]
                       CONF [CONF ...]
```

Positional Arguments

CONF The default configuration file(s) used to determine what base settings are unnecessary to keep in the target file.

Named Arguments

--target, -t The local file that you wish to remove duplicate settings from. By default, this file will be read from and then updated with a minimized version.

--dry-run, -D Enable dry-run mode. Instead of writing the minimizing the TARGET file, preview what would be removedthe form of a 'diff'.

--output Write the minimized output to a separate file instead of updating TARGET.

This option can be used to *preview* the actual changes. Sometimes if `--dry-run` mode produces too much output, it's helpful to look at the acutal minimized version of the file in concrete form (rather than a relative format, like a diff. This may also be helpful in other workflows.

--explode-default, -E Enable minimization across stanzas as well as files for special use-cases

This mode will not only minimize the same stanza across multiple config files, it will also attempt to minimize any default values stored in the `[default]` or global stanza as well. For this to be effective, it's often necessary to include system default in the CONF list. For example, to trim out cruft in savedsearches.conf, make sure you add `etc/system/default/savedsearches.conf` as an input.

-k, --preserve-key Specify attributes that should always be kept.

Example usage

```
cd Splunk_TA_nix
cp default/inputs.conf local/inputs.conf

# Edit 'disabled' and 'interval' settings in-place
vi local/inputs.conf

# Remove all the extra (unmodified) bits
ksconf minimize --target=local/inputs.conf default/inputs.conf
```

For special cases, the `--explode-default` mode reduces duplication between entries normal stanzas and global/default entries. If `disabled = 0` is a global default, it's technically safe to remove that setting from individual stanzas. But sometimes it's preferable to be explicit, and this behavior may be too heavy-handed for general use so it's off by default. Use this mode if you need your conf file that's been fully-expanded. (i.e., conf entries downloaded via REST, or the output of "btool list"). This isn't perfect, since many apps push their settings into the global namespace, but it can help. In many ways this process mimics what Splunk does *every* time it updates a conf file. The difference being that Splunk always has the full context, for this command to work most effectively, it really need to be given *all* the layers of default (system, app level, and so on). Also keep in mind that when ksconf load this formation, it isn't taking ACLs into consideration (the individual conf files are not linked to their metadata counterparts) so your results may vary from what a live Splunk system would do. Just something to think about. (Probably not a big deal since this is all on the fringes.)

3.4.8 ksconf promote

Propagate .conf settings applied in one file to another. Typically this is used to take local changes made via the UI and push them into a default (or default.d/) location.

NOTICE: By default, changes are *MOVED*, not just copied.

Promote has two different modes: batch and interactive. In batch mode all changes are applied automatically and the (now empty) source file is removed. In interactive mode the user is prompted to pick which stanzas and keys to integrate. This can be used to push changes made via the UI, which are stored in a 'local' file, to the version-controlled 'default' file. Note that the normal operation moves changes from the SOURCE file to the TARGET, updating both files in the process. But it's also possible to preserve the local file, if desired.

If either the source file or target file is modified while a promotion is under progress, changes will be aborted. And any custom selections you made will be lost. (This needs improvement.)

```
usage: ksconf promote [-h] [--batch | --interactive] [--force] [--keep]
                      [--keep-empty]
                      SOURCE TARGET
```

Positional Arguments

SOURCE	The source configuration file to pull changes from. Typically the 'local' conf file)
TARGET	Configuration file or directory to push the changes into. (Typically the 'default' folder) As a shortcut, a directory is given, then it's assumed that the same basename is used for both SOURCE and TARGET. In fact, if different basename as provided, a warning is issued.

Named Arguments

--batch, -b	Use batch mode where all configuration settings are automatically promoted. All changes are removed from source and applied to target. The source file will be removed, unless '–keep-empty' is used.
--interactive, -i	Enable interactive mode where the user will be prompted to approve the promotion of specific stanzas and keys. The user will be able to apply, skip, or edit the changes being promoted. (This functionality was inspired by 'git add –patch').
--force, -f	Disable safety checks.
--keep, -k	Keep conf settings in the source file. All changes will be copied into the target file instead of being moved there. This is typically a bad idea since local always overrides default.
--keep-empty	Keep the source file, even if after the settings promotions the file has no content. By default, SOURCE will be removed after all content has been moved into TARGET. Splunk will re-create any necessary local files on the fly.

3.4.9 ksconf rest-export

Build an executable script of the stanzas in a configuration file that can be later applied to a running Splunk instance via the Splunkd REST endpoint.

This can be helpful when pushing complex props & transforms to an instance where you only have UI access and can't directly publish an app.

```
usage: ksconf rest-export [-h] [--output FILE] [--disable-auth-output]
                          [--pretty-print] [-u | -D] [--url URL] [--app APP]
                          [--user USER] [--conf TYPE]
                          [--extra-args EXTRA_ARGS]
                          CONF [CONF ...]
```

Positional Arguments

CONF	Configuration file(s) to export settings from.

Named Arguments

--output, -t	Save the shell script output to this file. If not provided, the output is written to standard output.
-u, --update	Assume that the REST entities already exist. By default output assumes stanzas are being created. (This is an unfortunate quark of the configs REST API)
-D, --delete	Remove existing REST entities. This is a destructive operation. In this mode, stanzas attributes are unnecessary and ignored. NOTE: This works for 'local' entities only; the default folder cannot be updated.
--url	URL of Splunkd. Default: "https://localhost:8089"
--app	Set the namespace (app name) for the endpoint
--user	Set the user associated. Typically the default of 'nobody' is ideal if you want to share the configurations at the app-level.
--conf	Explicitly set the configuration file type. By default this is derived from CONF, but sometime it's helpful set this explicitly. Can be any valid Splunk conf file type, example include 'app', 'props', 'tags', 'savesdearches', and so on.
--extra-args	Extra arguments to pass to all CURL commands. Quote arguments on the commandline to prevent confusion between arguments to ksconf vs curl.

Output Control

--disable-auth-output	Turn off sample login curl commands from the output.
--pretty-print, -p	Enable pretty-printing. Make shell output a bit more readable by splitting entries across lines.

Warning: For interactive use only

This command is indented for manual admin workflows. It's quite possible that shell escaping bugs exist that may allow full shell access if you put this into an automated workflow. Evaluate the risks, review the code, and run as a least-privilege user, and be responsible.

Roadmap

For now the assumption is that `curl` command will be used. (Patches to support the Power Shell `Invoke-WebRequest` cmdlet would be greatly welcomed!)

Example

```
ksconf rest-export --output=apply_props.sh etc/app/Splunk_TA_aws/local/props.conf
```

3.4.10 ksconf snapshot

Build a static snapshot of various configuration files stored within a structured json export format. If the .conf files being captured are within a standard Splunk directory structure, then certain metadata is assumed based on path locations. Otherwise, less metadata is recorded.

ksconf snapshot –output=daily.json /opt/splunk/etc/app/

```
usage: ksconf snapshot [-h] [--output FILE] [--minimize] PATH [PATH ...]
```

Positional Arguments

PATH	Directory from which to load configuration files. All .conf and .meta file are included recursively.

Named Arguments

--output, -o	Save the snapshot to the named files. If not provided, the snapshot is written to standard output.
--minimize	Reduce the size of the JSON output by removing whitespace. Reduces readability.

Warning: Output NOT stable!

The output from this command hasn't really been tested in any kind of serious way for usability. Consider this a proof-of-concept. Anyone interested in this type of functionality should reach out and we can discuss uses cases.

3.4.11 ksconf sort

Sort a Splunk .conf file. Sort has two modes: (1) by default, the sorted config file will be echoed to the screen. (2) the config files are updated inplace when the -i' option is used.

Manually managed conf files can be blacklisted by add a comment containing the string `KSCONF-NO-SORT` to the top of any .conf file.

```
usage: ksconf sort [-h] [--target FILE | --inplace] [-F] [-q] [-n LINES]
                   FILE [FILE ...]
```

Positional Arguments

FILE	Input file to sort, or standard input.

Named Arguments

--target, -t	File to write results to. Defaults to standard output.
--inplace, -i	Replace the input file with a sorted version. Warning this a potentially destructive operation that may move/remove comments.
-n, --newlines	Lines between stanzas.

In-place update arguments

-F, --force	Force file sorting for all files, even for files containing the special 'KSCONF-NO-SORT' marker.
-q, --quiet	Reduce the output. Reports only updated or invalid files. This is useful for pre-commit hooks, for example.

Examples

To recursively sort all files

```
find . -name '*.conf' | xargs ksconf sort -i
```

3.4.12 ksconf unarchive

summary

Unarchive (or install) some splunk apps.

Install or overwrite an existing app in a git-friendly way. If the app already exist, steps will be taken to upgrade it safely.

The 'default' folder can be redirected to another path (i.e., 'default.d/10-upstream' or whatever which is helpful if you're using the ksconf 'combine' mode.)

```
usage: ksconf unarchive [-h] [--dest DIR] [--app-name NAME]
                        [--default-dir DIR] [--exclude EXCLUDE] [--keep KEEP]
                        [--allow-local]
                        [--git-sanity-check {off,changed,untracked,ignored}]
                        [--git-mode {nochange,stage,commit}] [--no-edit]
                        [--git-commit-args GIT_COMMIT_ARGS]
                        SPL
```

Positional Arguments

SPL The path to the archive to install.

Supports tarballs (.tar.gz, .spl), and less-common zip files (.zip)

Named Arguments

--dest Set the destination path where the archive will be extracted. By default the current directory is used, but sane values include etc/apps, etc/deployment-apps, and so on.

Often this will be a git repository working tree where splunk apps are stored.

--app-name The app name to use when expanding the archive. By default, the app name is taken from the archive as the top-level path included in the archive (by convention).

Expanding archives that contain multiple (ITSI) or nested apps (NIX, ES) is not supported.)

--default-dir Name of the directory where the default contents will be stored. This is a useful feature for apps that use a dynamic default directory that's created and managed by the 'combine' mode.

--exclude, -e Add a file pattern to exclude. Splunk's psudo-glob patterns are supported here. `*` for any non-directory match, `...` for ANY (including directories), and ? for a single character.

--keep, -k Specify a pattern for files to preserve during an upgrade. Repeat this argument to keep multiple patterns.

--allow-local Allow local/* and local.meta files to be extracted from the archive.

Shipping local files is a Splunk app packaging violation so local files are blocked to prevent content from being overridden.

--git-sanity-check By default `git status` is run on the destination folder to detect working tree or index modifications before the unarchive process starts, but this is configurable. Sanity check choices go from least restrictive to most thorough:

- Use `off` to prevent any 'git status' safely checks.
- Use `changed` to abort only upon local modifications to files tracked by git.
- Use `untracked` (the default) to look for changed and untracked files before considering the tree clean.
- Use `ignored` to enable the most intense safety check which will abort if local changes, untracked, or ignored files are found.

--git-mode Possible choices: nochange, stage, commit

Set the desired level of git integration. The default mode is *stage', where new, updated, or removed files are automatically handled for you.

To prevent any `git add` or `git rm` commands from being run, pick the 'nochange' mode.

If a git commit is incorrect, simply roll it back with `git reset` or fix it with a `git commit --amend` before the changes are pushed anywhere else. There's no native `--dry-run` or undo for unarchive mode because that's why you're using git in the first place, right? (And such features would require significant overhead and unittesting)

--no-edit Tell git to skip opening your editor. By default you will be prompted to review/edit the commit message. (Git Tip: Delete the content of the message to abort the commit.)

--git-commit-args, -G Extra arguments to pass to 'git'

Note: Git features are automatically disabled

Sanity checks and commit modes are automatically disabled if the app is being installed into a directory that is *not* a git working tree. And this check is only done after first confirming that git is present and functional.

3.5 Developer setup

The following steps highlight the developer install process.

3.5.1 Setup tools

If you are a developer then we strongly suggest installing into a virtual environment to prevent overwriting the production version of ksconf and for the installation of the developer tools. (The virtualenv name `ksconfdev-pyve` is used below, but this can be whatever suites, just make sure not to commit it.)

```
# Setup and activate virtual environment
virtualenv ksconfdev-pyve
. ksconfdev-pyve/bin/activate

# Install developer packages
pip install -r requirements-dev.txt
```

3.5.2 Install ksconf

```
git clone https://github.com/Kintyre/ksconf.git
cd ksconf
pip install .
```

3.5.3 Building the docs

```
cd ksconf
. ksconfdev-pyve/bin/activate

cd docs
make html
open build/html/index.html
```

If you'd like to build PDF, then you'll need some extra tools. On Mac, you may also want to install the following (for building docs, and the like):

```
brew install homebrew/cask/mactex-no-gui
```

3.6 Contributing

Pull requests are greatly welcome! If you plan on contributing code back to the main `ksconf` repo, please follow the standard GitHub fork and pull-request work-flow. We also ask that you enable a set of git hooks to help safeguard against avoidable issues.

3.6.1 Pre-commit hook

The ksconf project uses the pre-commit hook to enable the following checks:

- Fixes trailing whitespace, EOF, and EOLs
- Confirms python code compiles (AST)
- Blocks the committing of large files and keys
- Rebuilds the CLI docs. (Eventually to be replaced with an argparse Sphinx extension)
- Confirms that all Unit test pass. (Currently this is the same tests also run by Travis CI, but since test complete in under 5 seconds, the run-everywhere approach seems appropriate for now. Eventually, the local testing will likely become a subset of the full test suite.)

Note that this repo both uses pre-commit for it's own validation (as discussed here) and provides a pre-commit hook service to other repos. This way repositories housing Splunk apps can, for example, use `ksconf --check` or `ksconf --sort` against their own `.conf` files for validation purposes.

Installing the pre-commit hook

To run ensure you changes comply with the ksconf coding standards, please install and activate pre-commit.

Install:

```
sudo pip install pre-commit

# Register the pre-commit hooks (one time setup)
cd ksconf
pre-commit install --install-hooks
```

3.6.2 Install gitlint

Gitlint will check to ensure that commit messages are in compliance with the standard subject, empty-line, body format. You can enable it with:

```
gitlint install-hook
```

3.6.3 Refresh module listing

After making changes to the module hierarchy or simply adding new commands, refresh the listing for the autodoc extension by running the following command. Note that this may not remove old packages.

```
sphinx-apidoc -o docs/source/ ksconf --force
```

3.7 Git tips & tricks

3.7.1 Git configuration tweaks

Ksconf as external difftool

Use *ksconf diff* as an external *difftool* provider for **git**. Edit `~/.gitconfig` and add the following entices:

```
[difftool "ksconf"]
    cmd = "ksconf --force-color diff \"$LOCAL\" \"$REMOTE\" | less -R"
[difftool]
    prompt = false
[alias]
    ksdiff = "difftool --tool=ksconf"
```

Now you can run this new `git` alias to compare files in your directory using the `ksconf diff` feature instead of the default textual diff that git provides.

```
git ksdiff props.conf
```

Stanza aware textual diffs

Make `git diff` show the 'stanza' on the `@@` output lines.

Note: How does git know that?

Ever wonder how `git diff` is able to show you the name of the function or method where changes were made? This works for many programming languages out of the box. If you've ever spend much time looking at diffs that additional context is invaluable. As it turns out, this is customizable by adding a stanza matching regular expression with a file pattern match.

Simply add the following settings to your git configuration:

```
[diff "conf"]
    xfuncname = "^(\\[.*\\])$"
```

Then register this new ability with specific file patterns using git's `attributes` feature. Edit `~/.config/git/attributes` and add:

```
*.conf diff=conf
*.meta diff=conf
```

Note: Didn't work as expected?

Be aware that your location for your global-level attributes may be in a different location. In any case, you can use the following commands to test if the settings have been applied correctly.

```
git check-attr -a -- *.conf
```

Test to make sure the `xfuncname` attribute was set as expected:

```
git config diff.conf.xfuncname
```

3.8 Random

3.8.1 Typographic and Convention

Pronounced: k·s·knf

Capitalization:

Form	Acceptability factor
`ksconf`	Always lower for CLI. Generally preferred.
KSCONF	Okay for titles.
Ksconf	Title case is okay too.
KSConf	You'll see this, but weird.
KsConf	No, except maybe in a class name?
KsconF	Thought about it. Reserved for ASCII art ONLY

I wrote this while laughing at my own lack of consistency.

– Lowell

3.8.2 How Splunk writes to conf files

Splunk does some somewhat counter intuitive thing when it writes to local conf files.

For example,

1. All conf file updates are automatically minimized. (Splunk can get away with this because it *only* updates "local" files.)
2. Modified stanzas are removed from the current position in the .conf file and moved to the bottom.
3. Stanzas are typically re-written sorted in attribute order. (Or is it the same as #2? updated attributes are written to the bottom. *Note to editor: check on this*)
4. Sometimes boolean values persist in unexpected ways. (Primarily this is because there's mor than one way to represent them textually, and that textual representation is different from what's stored in default)

Essentially, splunk will always "minimize" the conf file at each any every update. This is because Splunk internally keeps track of the final representation of the entire stanza (in memory), and only when it's written to disk does Splunk care about the the current contents of the local file. In fact, Splunk re-reads the conf file immediately before updating it. This is why, if you've made a local changes, and forgot to reload, Splunk will typically not lose your change (unless you've update the same attribute both places… I mean, it's not magic.)

Tip: Don't believe me? Try it yourself.

To prove that it works this way, simply find a savedsearch that you modified from any app that you installed. Look at the local conf file and observe your changes. Now go edit the saved search and restore some attribute to it's original value (the most obvious one here would be the `search` attribute), but that's tricky if it's multiple lines. Now go look at the local conf file again. If you updated it with *exactly* the same value, then that attribute will have been completely removed from the local file. This is in fact a neat trick that can be used to revert local changes to allow future updates to "pass-though" unimpeded. In SHC scenarios, this may be your only option to remove local settings.

Okay, so what's the value in having a *minimize* command if Splunk does this automatically every time it's makes a change? Well, simply put, because Splunk can't write to all local file locations. Splunk only writes to system, etc/users, and etc/apps local folders (and sometimes to deployment-apps app.conf local file, but that's a completely different story.)

Also, there's also times where boolean values will show up in an unexpected manor because of how Splunk treats them internally. I'm still not sure if this is a silly mistake in the default .conf files or a clever workaround to what's essentially a design flaw in the conf system. But either way, I suspect the user benefits. Because splunk accepts more values as boolean than what it will write out, this means that certain boolean values will always be explicitly store in the conf files. This means that man `disabled` and bunches of other settings in `savedsearches.conf` always get explicitly written. How is that helpful? Well, imagine what would happen if you accidentally changed `disabled = 1` in the global stanzas in savedsearches.conf. Well, *nothing* if all savedsearches have that values explicitly written. The point is this: there are times when repeating yourself isn't a bad thing. (Incidentally, this is the reason for the `--preserve-key` flag on the *minimize* command.)

3.8.3 Grandfather Paradox

The KSCONF Splunk app breaks it's designed paradigm (not in a good way). Ksconf was designed to be the thing that manages all your other apps, so by deploying ksconf as an app itself, we open up the possibility that ksconf could upgrade it self or deploy itself, or manage itself. Basically it could cut off the limb that it's standing on. So practically this can get messy, especially if you're on Windows where file locking is also likely to cause issues for you.

So sure, if you want to be picky, "Grandfather paradox" is probably the wrong analogy. Pull requests welcome.

3.9 Command line reference

KSCONF supports the following CLI options:

3.9.1 ksconf

```
usage: ksconf [-h] [--version] [--force-color]
              {check,combine,diff,filter,promote,merge,minimize,snapshot,sort,rest-
↪export,unarchive}
              ...

Ksconf: Kintyre Splunk CONFig tool

This utility handles a number of common Splunk app maintenance tasks in a small
and easy to deploy package.  Specifically, this tools deals with many of the
nuances with storing Splunk apps in git, and pointing live Splunk apps to a git
repository.  Merging changes from the live system's (local) folder to the
version controlled (default) folder, and dealing with more than one layer of
"default" (which splunk can't handle natively) are all supported tasks.
```

(continues on next page)

(continued from previous page)

```
positional arguments:
  {check,combine,diff,filter,promote,merge,minimize,snapshot,sort,rest-export,
↪unarchive}
    check               Perform basic syntax and sanity checks on .conf files
    combine             Combine configuration files across multiple source
                        directories into a single destination directory. This
                        allows for an arbitrary number of splunk configuration
                        layers to coexist within a single app. Useful in both
                        ongoing merge and one-time ad-hoc use.
    diff                Compare settings differences between two .conf files
                        ignoring spacing and sort order
    filter              A stanza-aware GREP tool for conf files
    promote             Promote .conf settings from one file into another
                        either in batch mode (all changes) or interactively
                        allowing the user to pick which stanzas and keys to
                        integrate. Changes made via the UI (stored in the
                        local folder) can be promoted (moved) to a version-
                        controlled directory.
    merge               Merge two or more .conf files
    minimize            Minimize the target file by removing entries
                        duplicated in the default conf(s)
    snapshot            Snapshot .conf file directories into a JSON dump
                        format
    sort                Sort a Splunk .conf file creating a normalized format
                        appropriate for version control
    rest-export         Export .conf settings as a curl script to apply to a
                        Splunk instance later (via REST)
    unarchive           Install or upgrade an existing app in a git-friendly
                        and safe way

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  --force-color         Force TTY color mode on. Useful if piping the output a
                        color-aware pager, like 'less -R'
```

3.9.2 ksconf check

```
usage: ksconf check [-h] [--quiet] FILE [FILE ...]

Provide basic syntax and sanity checking for Splunk's .conf files. Use
Splunk's builtin 'btool check' for a more robust validation of keys and
values. Consider using this utility as part of a pre-commit hook.

positional arguments:
  FILE        One or more configuration files to check. If '-' is given, then
              read a list of files to validate from standard input

optional arguments:
```

(continues on next page)

(continued from previous page)

```
  -h, --help   show this help message and exit
  --quiet, -q  Reduce the volume of output.
```

3.9.3 ksconf combine

```
usage: ksconf combine [-h] [--target TARGET] [--dry-run] [--banner BANNER]
                      source [source ...]

Merge .conf settings from multiple source directories into a combined target
directory.   Configuration files can be stored in a '/etc/*.d' like directory
structure and consolidated back into a single 'default' directory.

This command supports both one-time operations and recurring merge jobs.  For
example, this command can be used to combine all users knowledge objects (stored
in 'etc/users') after a server migration, or to merge a single user's settings
after an their account has been renamed.  Recurring operations assume some type
of external scheduler is being used.  A best-effort is made to only write to
target files as needed.

The 'combine' command takes your logical layers of configs (upstream, corporate,
splunk admin fixes, and power user knowledge objects, ...) expressed as
individual folders and merges them all back into the single 'default' folder
that Splunk reads from.  One way to keep the 'default' folder up-to-date is
using client-side git hooks.

No directory layout is mandatory, but but one simple approach is to model your
layers using a prioritized 'default.d' directory structure. (This idea is
borrowed from the Unix System V concept where many services natively read their
config files from '/etc/*.d' directories.)

positional arguments:
  source                The source directory where configuration files will be
                        merged from. When multiple sources directories are
                        provided, start with the most general and end with the
                        specific; later sources will override values from the
                        earlier ones. Supports wildcards so a typical Unix
                        'conf.d/##-NAME' directory structure works well.

optional arguments:
  -h, --help            show this help message and exit
  --target TARGET, -t TARGET
                        Directory where the merged files will be stored.
                        Typically either 'default' or 'local'
  --dry-run, -D         Enable dry-run mode. Instead of writing to TARGET,
                        preview changes as a 'diff'. If TARGET doesn't exist,
                        then show the merged file.
  --banner BANNER, -b BANNER
                        A warning banner to discourage manual editing of conf
                        files.
```

3.9.4 ksconf diff

```
usage: ksconf diff [-h] [-o FILE] [--comments] CONF1 CONF2

Compares the content differences of two .conf files

This command ignores textual differences (like order, spacing, and comments) and
focuses strictly on comparing stanzas, keys, and values.  Note that spaces
within any given value will be compared.  Multiline fields are compared in are
compared in a more traditional 'diff' output so that long savedsearches and
macros can be compared more easily.

positional arguments:
  CONF1                 Left side of the comparison
  CONF2                 Right side of the comparison

optional arguments:
  -h, --help            show this help message and exit
  -o FILE, --output FILE
                        File where difference is stored. Defaults to standard
                        out.
  --comments, -C        Enable comparison of comments. (Unlikely to work
                        consistently)
```

3.9.5 ksconf filter

```
usage: ksconf filter [-h] [-o FILE] [--comments] [--verbose]
                     [--match {regex,wildcard,string}] [--ignore-case]
                     [--invert-match] [--files-with-matches]
                     [--count | --brief] [--stanza PATTERN]
                     [--attr-present ATTR] [--keep-attrs WC-ATTR]
                     [--reject-attrs WC-ATTR]
                     CONF [CONF ...]

Filter the contents of a conf file in various ways. Stanzas can be included or
excluded based on provided filter, based on the presents or value of a key.
Where possible, this command supports GREP-like arguments to bring a familiar
feel.

positional arguments:
  CONF                  Input conf file

optional arguments:
  -h, --help            show this help message and exit
  -o FILE, --output FILE
                        File where the filtered results are written. Defaults
                        to standard out.
  --comments, -C        Preserve comments. Comments are discarded by default.
  --verbose             Enable additional output.
  --match {regex,wildcard,string}, -m {regex,wildcard,string}
```

(continues on next page)

(continued from previous page)

```
                        Specify pattern matching mode. Defaults to 'wildcard'
                        allowing for '*' and '?' matching. Use 'regex' for
                        more power but watch out for shell escaping. Use
                        'string' enable literal matching.
  --ignore-case, -i     Ignore case when comparing or matching strings. By
                        default matches are case-sensitive.
  --invert-match, -v    Invert match results. This can be used to show what
                        content does NOT match, or make a backup copy of
                        excluded content.

Output mode:
  Select an alternate output mode. If any of the following options are used,
  the stanza output is not shown.

  --files-with-matches, -l
                        List files that match the given search criteria
  --count, -c           Count matching stanzas
  --brief, -b           List name of matching stanzas

Stanza selection:
  Include or exclude entire stanzas using these filter options. All filter
  options can be provided multiple times. If you have a long list of
  filters, they can be saved in a file and referenced using the special
  'file://' prefix.

  --stanza PATTERN      Match any stanza who's name matches the given pattern.
                        PATTERN supports bulk patterns via the 'file://'
                        prefix.
  --attr-present ATTR   Match any stanza that includes the ATTR attribute.
                        ATTR supports bulk attribute patterns via the
                        'file://' prefix.

Attribute selection:
  Include or exclude attributes passed through. By default all attributes
  are preserved. Whitelist (keep) operations are preformed before blacklist
  (reject) operations.

  --keep-attrs WC-ATTR  Select which attribute(s) will be preserved. This
                        space separated list of attributes indicates what to
                        preserve. Supports wildcards.
  --reject-attrs WC-ATTR
                        Select which attribute(s) will be discarded. This
                        space separated list of attributes indicates what to
                        discard. Supports wildcards.
```

3.9.6 ksconf promote

```
usage: ksconf promote [-h] [--batch | --interactive] [--force] [--keep]
                      [--keep-empty]
                      SOURCE TARGET
```

(continues on next page)

(continued from previous page)

```

Propagate .conf settings applied in one file to another.  Typically this is used
to take local changes made via the UI and push them into a default (or
default.d/) location.

NOTICE:  By default, changes are *MOVED*, not just copied.

Promote has two different modes:  batch and interactive.  In batch mode all
changes are applied automatically and the (now empty) source file is removed.
In interactive mode the user is prompted to pick which stanzas and keys to
integrate.  This can be used to push  changes made via the UI, which are stored
in a 'local' file, to the version-controlled 'default' file.  Note that the
normal operation moves changes from the SOURCE file to the TARGET, updating both
files in the process.  But it's also possible to preserve the local file, if
desired.

If either the source file or target file is modified while a promotion is under
progress, changes will be aborted.  And any custom selections you made will be
lost.  (This needs improvement.)

positional arguments:
  SOURCE              The source configuration file to pull changes from.
                      Typically the 'local' conf file)
  TARGET              Configuration file or directory to push the changes into.
                      (Typically the 'default' folder) As a shortcut, a
                      directory is given, then it's assumed that the same
                      basename is used for both SOURCE and TARGET. In fact, if
                      different basename as provided, a warning is issued.

optional arguments:
  -h, --help          show this help message and exit
  --batch, -b         Use batch mode where all configuration settings are
                      automatically promoted. All changes are removed from
                      source and applied to target. The source file will be
                      removed, unless '--keep-empty' is used.
  --interactive, -i   Enable interactive mode where the user will be prompted
                      to approve the promotion of specific stanzas and keys.
                      The user will be able to apply, skip, or edit the changes
                      being promoted. (This functionality was inspired by 'git
                      add --patch').
  --force, -f         Disable safety checks.
  --keep, -k          Keep conf settings in the source file. All changes will
                      be copied into the target file instead of being moved
                      there. This is typically a bad idea since local always
                      overrides default.
  --keep-empty        Keep the source file, even if after the settings
                      promotions the file has no content. By default, SOURCE
                      will be removed after all content has been moved into
                      TARGET. Splunk will re-create any necessary local files
                      on the fly.
```

3.9.7 ksconf merge

```
usage: ksconf merge [-h] [--target FILE] [--dry-run] [--banner BANNER]
                    FILE [FILE ...]

Merge two or more .conf files into a single combined .conf file.  This could be
used to merge the props.conf file from ALL technology addons into a single file:

ksconf merge --target=all-ta-props.conf etc/apps/*TA*/{default,local}/props.conf

positional arguments:
  FILE                  The source configuration file to pull changes from.

optional arguments:
  -h, --help            show this help message and exit
  --target FILE, -t FILE
                        Save the merged configuration files to this target
                        file. If not provided. the the merged conf is written
                        to standard output.
  --dry-run, -D         Enable dry-run mode. Instead of writing to TARGET,
                        preview changes in 'diff' format. If TARGET doesn't
                        exist, then show the merged file.
  --banner BANNER, -b BANNER
                        A banner or warning comment added to the top of the
                        TARGET file. This is often used to warn Splunk admins
                        from editing an auto-generated file.
```

3.9.8 ksconf minimize

```
usage: ksconf minimize [-h] [--target TARGET] [--dry-run | --output OUTPUT]
                       [--explode-default] [-k PRESERVE_KEY]
                       CONF [CONF ...]

Minimize a conf file by removing the default settings

Reduce local conf file to only your indented changes without manually tracking
which entries you've edited.  Minimizing local conf files makes your local
customizations easier to read and often results in cleaner add-on upgrades.

positional arguments:
  CONF                  The default configuration file(s) used to determine
                        what base settings are unnecessary to keep in the
                        target file.

optional arguments:
  -h, --help            show this help message and exit
  --target TARGET, -t TARGET
                        The local file that you wish to remove duplicate
                        settings from. By default, this file will be read from
                        and then updated with a minimized version.
```

(continues on next page)

(continued from previous page)

```
  --dry-run, -D         Enable dry-run mode. Instead of writing the minimizing
                        the TARGET file, preview what would be removedthe form
                        of a 'diff'.
  --output OUTPUT       Write the minimized output to a separate file instead
                        of updating TARGET.
  --explode-default, -E
                        Enable minimization across stanzas as well as files
                        for special use-cases
  -k PRESERVE_KEY, --preserve-key PRESERVE_KEY
                        Specify attributes that should always be kept.
```

3.9.9 ksconf snapshot

```
usage: ksconf snapshot [-h] [--output FILE] [--minimize] PATH [PATH ...]

Build a static snapshot of various configuration files stored within a
structured json export format. If the .conf files being captured are within a
standard Splunk directory structure, then certain metadata is assumed based on
path locations. Otherwise, less metadata is recorded. ksconf snapshot
--output=daily.json /opt/splunk/etc/app/

positional arguments:
  PATH                  Directory from which to load configuration files. All
                        .conf and .meta file are included recursively.

optional arguments:
  -h, --help            show this help message and exit
  --output FILE, -o FILE
                        Save the snapshot to the named files. If not provided,
                        the snapshot is written to standard output.
  --minimize            Reduce the size of the JSON output by removing
                        whitespace. Reduces readability.
```

3.9.10 ksconf sort

```
usage: ksconf sort [-h] [--target FILE | --inplace] [-F] [-q] [-n LINES]
                   FILE [FILE ...]

Sort a Splunk .conf file.  Sort has two modes:  (1) by default, the sorted
config file will be echoed to the screen.  (2) the config files are updated
inplace when the -i' option is used.

Manually managed conf files can be blacklisted by add a comment containing the
string 'KSCONF-NO-SORT' to the top of any .conf file.

positional arguments:
  FILE                  Input file to sort, or standard input.
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  --target FILE, -t FILE
                        File to write results to. Defaults to standard output.
  --inplace, -i         Replace the input file with a sorted version. Warning
                        this a potentially destructive operation that may
                        move/remove comments.
  -n LINES, --newlines LINES
                        Lines between stanzas.

In-place update arguments:
  -F, --force           Force file sorting for all files, even for files
                        containing the special 'KSCONF-NO-SORT' marker.
  -q, --quiet           Reduce the output. Reports only updated or invalid
                        files. This is useful for pre-commit hooks, for
                        example.
```

3.9.11 ksconf rest-export

```
usage: ksconf rest-export [-h] [--output FILE] [--disable-auth-output]
                          [--pretty-print] [-u | -D] [--url URL] [--app APP]
                          [--user USER] [--conf TYPE]
                          [--extra-args EXTRA_ARGS]
                          CONF [CONF ...]

Build an executable script of the stanzas in a configuration file that can be␣
↪later applied to
a running Splunk instance via the Splunkd REST endpoint.

This can be helpful when pushing complex props & transforms to an instance where␣
↪you only have
UI access and can't directly publish an app.

positional arguments:
  CONF                  Configuration file(s) to export settings from.

optional arguments:
  -h, --help            show this help message and exit
  --output FILE, -t FILE
                        Save the shell script output to this file. If not
                        provided, the output is written to standard output.
  -u, --update          Assume that the REST entities already exist. By
                        default output assumes stanzas are being created.
                        (This is an unfortunate quark of the configs REST API)
  -D, --delete          Remove existing REST entities. This is a destructive
                        operation. In this mode, stanzas attributes are
                        unnecessary and ignored. NOTE: This works for 'local'
                        entities only; the default folder cannot be updated.
  --url URL             URL of Splunkd. Default: https://localhost:8089
  --app APP             Set the namespace (app name) for the endpoint
```

(continues on next page)

(continued from previous page)

```
  --user USER           Set the user associated. Typically the default of
                        'nobody' is ideal if you want to share the
                        configurations at the app-level.
  --conf TYPE           Explicitly set the configuration file type. By default
                        this is derived from CONF, but sometime it's helpful
                        set this explicitly. Can be any valid Splunk conf file
                        type, example include 'app', 'props', 'tags',
                        'savesdearches', and so on.
  --extra-args EXTRA_ARGS
                        Extra arguments to pass to all CURL commands. Quote
                        arguments on the commandline to prevent confusion
                        between arguments to ksconf vs curl.

Output Control:
  --disable-auth-output
                        Turn off sample login curl commands from the output.
  --pretty-print, -p    Enable pretty-printing. Make shell output a bit more
                        readable by splitting entries across lines.
```

3.9.12 ksconf unarchive

```
usage: ksconf unarchive [-h] [--dest DIR] [--app-name NAME]
                        [--default-dir DIR] [--exclude EXCLUDE] [--keep KEEP]
                        [--allow-local]
                        [--git-sanity-check {off,changed,untracked,ignored}]
                        [--git-mode {nochange,stage,commit}] [--no-edit]
                        [--git-commit-args GIT_COMMIT_ARGS]
                        SPL

Install or overwrite an existing app in a git-friendly way.
If the app already exist, steps will be taken to upgrade it safely.

The 'default' folder can be redirected to another path (i.e., 'default.d/10-
↪upstream' or
whatever which is helpful if you're using the ksconf 'combine' mode.)

positional arguments:
  SPL                   The path to the archive to install.

optional arguments:
  -h, --help            show this help message and exit
  --dest DIR            Set the destination path where the archive will be
                        extracted. By default the current directory is used,
                        but sane values include etc/apps, etc/deployment-apps,
                        and so on.
  --app-name NAME       The app name to use when expanding the archive. By
                        default, the app name is taken from the archive as the
                        top-level path included in the archive (by
                        convention).
  --default-dir DIR     Name of the directory where the default contents will
```

(continues on next page)

(continued from previous page)

```
                      be stored. This is a useful feature for apps that use
                      a dynamic default directory that's created and managed
                      by the 'combine' mode.
--exclude EXCLUDE, -e EXCLUDE
                      Add a file pattern to exclude. Splunk's psudo-glob
                      patterns are supported here. '*' for any non-directory
                      match, '...' for ANY (including directories), and '?'
                      for a single character.
--keep KEEP, -k KEEP  Specify a pattern for files to preserve during an
                      upgrade. Repeat this argument to keep multiple
                      patterns.
--allow-local         Allow local/* and local.meta files to be extracted
                      from the archive.
--git-sanity-check {off,changed,untracked,ignored}
                      By default 'git status' is run on the destination
                      folder to detect working tree or index modifications
                      before the unarchive process start. Sanity check
                      choices go from least restrictive to most thorough:
                      'off' prevents all safely checks. 'changed' aborts
                      only upon local modifications to files tracked by git.
                      'untracked' (the default) looks for changed and
                      untracked files. 'ignored' aborts is (any) local
                      changes, untracked, or ignored files are found.
--git-mode {nochange,stage,commit}
                      Set the desired level of git integration. The default
                      mode is *stage', where new, updated, or removed files
                      are automatically handled for you. To prevent any 'git
                      add' or 'git rm' commands from being run, pick the
                      'nochange' mode.
--no-edit             Tell git to skip opening your editor. By default you
                      will be prompted to review/edit the commit message.
                      (Git Tip: Delete the content of the message to abort
                      the commit.)
--git-commit-args GIT_COMMIT_ARGS, -G GIT_COMMIT_ARGS
                      Extra arguments to pass to 'git'
```

3.10 Changelog

Note: Changes in master, but not released yet are marked as *DRAFT*.

3.10.1 Ksconf 0.6.x

Add deployment as a Splunk app for simplicity and significant docs cleanup.

Release v0.6.2 (2019-02-09)

- Massive rewrite and restructuring of the docs. Highlights include:
  - Reference material has been moved out of the user manual into a different top-level section.
  - Many new topics were added, such as
    - *Ksconf as external difftool*
    - *How Splunk writes to conf files*
    - *Configuration layers*
    - *What's so important about minimizing files?*
  - A new approach for CLI documentation. We're moving away from the **WALL OF TEXT** thing. (Yeah, it was really just the output from `--help`). That was limiting formatting, linking, and making the CLI output way too long.
- Refreshed Splunk app icons. Add missing alt icon.
- Several minor internal cleanups. Specifically the output of `--version` had a face lift.

Release v0.6.1 (2019-02-07)

- (Trivial) Fixed some small issues with the Splunk App (online AppInspect)

Release v0.6.0 (2019-02-06)

- Add initial support for building ksconf into a Splunk app.
  - App contains a local copy of the docs, helpful for anyone who's working offline.
  - Credit to Sarah Larson for the ksconf logos.
  - No `ksconf` functionality exposed to the Splunk UI at the moment.
- Docs/Sphinx improvements (more coming)
  - Begin work on cleaning up API docs.
  - Started converting various document pages into reStructuredText for greatly improved docs.
  - Improved PDF fonts and fixed a bunch of sphinx errors/warnings.
- Refactored the install docs into 2 parts. With the new ability to install ksconf as a Splunk app it's quite likely that most of the wonky corner cases will be less frequently needed, hence all the more exotic content was moved into the "Advanced Install Guide", tidying things up.

3.10.2 Ksconf 0.5.x

Add Python 3 support, new commands, support for external command plugins, tox and vagrant for testing.

Release v0.5.6 (2019-02-04)

- Fixes and improvements to the `filter` command. Found issue with processing from stdin, inconsistency in some CLI arguments, and finished implementation for various output modes.
- Add logo (fist attempt).

Release v0.5.5 (2019-01-28)

- New *ksconf filter* command added for slicing up a conf file into smaller pieces. Think of this as GREP that's stanza-aware. Can also whitelist or blacklist attributes, if desirable.
- Expanded `rest-export` CLI capabilities to include a new `delete` option, pretty-printing, and now supports stdin by allowing the user to explicitly set the file type using `conf`.
- Refactored all CLI unittests for increased readability and long-term maintenance. Unit tests now can also be run individually as scripts from the command line.
- Minor tweaks to the `snapshot` output format, v0.2. This feature is still highly experimental.

Release v0.5.4 (2019-01-04)

- New commands added:
  - *ksconf snapshot* will dump a set of configuration files to a JSON formatted file. This can be used used for incremental "snapshotting" of running Splunk apps to track changes overtime.
  - *ksconf rest-export* builds a series of custom `curl` commands that can be used to publish or update stanzas on a remote instance without file system access. This can be helpful when pushing configs to Splunk Cloud when all you have is REST (splunkd) access. This command is indented for interactive admin not batch operations.
- Added the concept of command maturity. A listing is available by running `ksconf --version`
- Fix typo in `KSCONF_DEBUG`.
- Resolving some build issues.
- Improved support for development/testing environments using Vagrant (fixes) and Docker (new). Thanks to Lars Jonsson for these enhancements.

Release v0.5.3 (2018-11-02)

- Fixed bug where `ksconf combine` could incorrectly order directories on certain file systems (like ext4), effectively ignoring priorities. Repeated runs may resulted in undefined behavior. Solved by explicitly sorting input paths forcing processing to be done in lexicographical order.
- Fixed more issues with handling files with BOM encodings. BOMs and encodings in general are NOT preserved by ksconf. If this is an issue for you, please add an enhancement issue.
- Add Python 3.7 support
- Expand install docs specifically for offline mode and some OS-specific notes.
- Enable additional tracebacks for CLI debugging by setting `KSCONF_DEBUG=1` in the environment.

Release v0.5.2 (2018-08-13)

- Expand CLI output for `--help` and `--version`
- Internal cleanup of CLI entry point module name. Now the ksconf CLI can be invoked as `python -m ksconf`, you know, for anyone who's into that sort of thing.
- Minor docs and CI/testing improvements.

Release v0.5.1 (2018-06-28)

- Support external ksconf command plugins through custom *entry_points*, allowing for others to develop their own custom extensions as needed.
- Many internal changes: Refactoring of all CLI commands to use new entry_points as well as pave the way for future CLI unittest improvements.
- Docs cleanup / improvements.

Release v0.5.0 (2018-06-26)

- Python 3 support.
- Many bug fixes and improvements resulting from wider testing.

3.10.3 Ksconf 0.4.x

Ksconf 0.4.x switched to a modular code base, added build/release automation, PyPI package registration (installation via `pip install` and, online docs.

Release v0.4.10 (2018-06-26)

- Improve file handling to avoid "unclosed file" warnings. Impacted `parse_conf()`, `write_conf()`, and many unittest helpers.
- Update badges to report on the master branch only. (No need to highlight failures on feature or bug-fix branches.)

Release v0.4.9 (2018-06-05)

- Add some missing docs files

Release v0.4.8 (2018-06-05)

- Massive cleanup of docs: revamped install guide, added 'standalone' install procedure and developer-focused docs. Updated license handling.
- Updated docs configuration to dynamically pull in the ksconf version number.
- Using the classic 'read-the-docs' Sphinx theme.
- Added additional PyPi badges to README (GitHub home page).

Release v0.4.4-v0.4.1 (2018-06-04)

- Deployment and install fixes (It's difficult to troubleshoot/test without making a new release!)

Release v0.4.3 (2018-06-04)

- Rename PyPI package `kintyre-splunk-conf`
- Add support for building a standalone executable (zipapp).
- Revamp install docs and location
- Add GitHub release for the standalone executable.

Release v0.4.2 (2018-06-04)

- Add readthedocs.io support

Release v0.4.1 (2018-06-04)

- Enable PyPI production package building

Release v0.4.0 (2018-05-19)

- Refactor entire code base. Switched from monolithic all-in-one file to clean-cut modules.
- Versioning is now discoverable via `ksconf --version`, and controlled via git tags (via `git describe --tags`).

Module layout

- `ksconf.conf.*` - Configuration file parsing, writing, comparing, and so on
- `ksconf.util.*` - Various helper functions
- `ksconf.archive` - Support for uncompressing Splunk apps (tgz/zip files)
- `ksconf.vc.git` - Version control support. Git is the only VC tool supported for now. (Possibly ever)
- `ksconf.commands.<CMD>` - Modules for specific CLI functions. I may make this extendable, eventually.

3.10.4 Ksconf 0.3.x

First public releases.

Release v0.3.2 (2018-04-24)

- Add AppVeyor for Windows platform testing
- Add codecov integration
- Created ConfFileProxy.dump()

Release v0.3.1 (2018-04-21)

- Setup automation via Travis CI
- Add code coverage

Release v0.3.0 (2018-04-21)

- Switched to semantic versioning.
- 0.3.0 feels representative of the code maturity.

3.10.5 Ksconf legacy releases

Ksconf started in a private Kintyre repo. There are no official releases; all git history has been rewritten.

Release legacy-v1.0.1 (2018-04-20)

- Fixes to blacklist support and many enhancements to `ksconf unarchive`.
- Introduces parsing profiles.
- Lots of bug fixes to various subcommands.
- Added automatic detection of 'subcommands' for CLI documentation helper script.

Release legacy-v1.0.0 (2018-04-16)

- This is the first public release. First work began Nov 2017 (as a simple conf 'sort' tool, which was imported from yet another repo.) Version history was extracted/rewritten/preserved as much as possible.
- Mostly stable features.
- Unit test coverage over 85%
- Includes pre-commit hook configuration (so that other repos can use this to run `ksconf sort` and `ksconf check` against their conf files.

3.11 Known issues

3.11.1 General

- File encoding issues:
- Byte order markers and specific encodings are NOT preserved. All file will be writen out as UTF-8, by default.

3.11.2 Splunk app

- File cleanup issues after *KSCONF app for Splunk* upgrades. This can be caused by old *.dist-info* folders or other stale files left around from between versions. If you encounter this issue, either uninstall (and, if necessary wipe the directory) or just manually remove the old dist-info folders. A proper solution for this is needed. (GH issue #37)

See more confirmed bugs in the issue tracker.

3.12 Advanced Installation Guide

The content in this document was split out from the *Installation Guide* because it became unruly and the number of possible Python installation combinations and gotchas became very intense. However, that means that there's lots of truly helpful stuff in here, but becoming a python packaging expert isn't my goal, so the Splunk app install approach was introduced to alleviate much of this pain.

A portion of this document is targeted at those who can't install packages as Admin or are forced to use Splunk's embedded Python. For everyone else, please start with the one-liner!

Tip: Do any of these words for phrases strike fear in your heart?

- `pip`
- `pipenv`
- `virtualenv`
- `wheel`
- `pyenv` (not the same as `pyvenv`)
- `python2.7` vs `python27` vs `py -27`
- `PYTHONPATH`
- `LD_LIBARY`
- RedHat Software Collections

If this list seems daungting, head over to *Install Splunk App*. There's no shame in it.

Contents

3.12.1 Flowchart

(Unfinished; more of a brainstorm at this point. . .)

- Is Python installed? (OS level)
  - Is the version greater than 2.7? (Some early 2.7 version have quarks, but typically this is okay)
  - If Python 3.x, is it greater than 3.4? (I'd like to drop 3.4, but lots of old distros still have it.)
- Do you have admin access? (root/Administrator; or can you get it? How hard? Will you need it each time you upgrade the ksconf?)
- Do you already have a large python deployment or dependency? (If so, you'll probably be fine. Use virtualenv)
- Do you have any prior Python packaging or administration experience?
- Are you dealing with some vendor-specific solution?
  - Example: RedHat Software Collections – where they realize there software is way too old, so they try to make it possible to install newever version of things like Python, but since they aren't native or the default, you still end up jumping through a bunch of wonky hoops)
- Do you have Internet connectivity? (air gap or blocked outbound traffic, or proxy)
- Do you want to build/deploy your own ksconf extensions? If so, the python package is a better option. (But at that point, you can probably already handle any packaging issues yourself.)

3.12.2 Installation

There are several ways to install ksconf. Technically all standard python packaging approaches should work just fine, there's no compiled code or external run-time dependencies so installation is fairly easy, but for non-python developers there are some gotchas. Installation options are listed from the most easy and recommended to more obscure and difficult:

Install from PyPI with PIP

The preferred installation method is to install via the standard Python package tool **pip**. Ksconf can be installed via the registered kintyre-splunk-conf package using the standard python process.

There are 2 popular variations, depending on whether or not you would like to install for all users or just play around with it locally.

Install ksconf into a virtual environment

Use this option if you don't have admin access

Installing `ksconf` with virtualenv is a great way to test the tool without requiring admin privileges and has many advantages for a production install too. Here are the basic steps to get started.

Please change `venv` to a suitable path for your environment.

```
# Install Python virtualenv package (if not already installed)
pip install virtualenv

# Create and activte new 'venv' virtual environment
virtualenv venv
source venv/bin/activate

pip install kintyre-splunk-conf
```

Note: Windows users

The above virtual environment activation should be run as `venvScriptsactivate.bat`.

Install ksconf system-wide

Important: This requires admin access.

This is the absolute easiest install method where 'ksconf' is available to all users on the system but it requires root access and `pip` must be installed and up-to-date.

On Mac or Linux, run:

```
sudo pip install kintyre-splunk-conf
```

On Windows, run this commands from an Administrator console.

```
pip install kintyre-splunk-conf
```

CentOS (RedHat derived) distros

```
# Enable the EPEL repo so that `pip` can be installed.
sudo yum install -y epel-release

# Install pip
sudo yum install -y python-pip

# Install ksconf (globally, for all users)
sudo pip install kintyre-splunk-conf
```

RedHat Software Collections

The following assumes the `python27` software collection, but other version of Python are supported too. The initial setup and deployment of Software Collections is beyond the scope of this doc.

```
sudo scl enable python27 python -m pip install kintyre-splunk-conf
```

Hint: Missing pip?

If pip is missing from a RHSC then install the following rpm.

```
yum install python27-python-pip
```

Unfortunately, the `ksconf` entrypoint script (in the `bin` folder) will not work correctly on it's own because it doesn't know about the scl environment, nor is it in the default PATH. To solve this run the following:

```
sudo cat > /usr/local/bin/ksconf <<HERE
#!/bin/sh
source scl_source enable python27
exec /opt/rh/python27/root/usr/bin/ksconf "$@"
HERE
chmod +x /usr/local/bin/ksconf
```

3.12.3 Use the standalone executable

Deprecated since version 0.6.0: This option remains for historical reference and will like be disabled in the future. If this seems like the best option to you, then please consider install the KSCONF App for Splunk instead.

Ksconf can be installed as a standalone executable zip app. This approach still requires a python interpreter to be present either from the OS or the one embedded with Splunk Enterprise. This works well for testing or when all other options fail.

From the GitHub releases page, grab the file name `ksconf-*.pyz`, download it, copy it to a `bin` folder in your PATH and rename it `ksconf`. The default shebang looks for 'python' in the PATH, but this can be adjusted as needed. Since installing with Splunk is a common use case, a second file named `ksconf-*-splunk.pyz` already has the shebang set for the standard `/opt/splunk` install path.

Typical embedded Splunk install example:

```
VER=0.5.0
curl https://github.com/Kintyre/ksconf/releases/download/v${VER}/ksconf-${VER}-splunk.pyz
mv ksconf-${VER}-splunk.pyz /opt/splunk/bin/
cd /opt/splunk/bin
ln -sf ksconf-${VER}-splunk.pyz ksconf
chmod +x ksconf
ksconf --version
```

Reasons why this is a non-ideal install approach:

- Lower performance since all python files live in a zip file, and precompiled version's can be cached.
- No standard install pathway (doesn't use pip); user must manually copy the executable into place.
- Uses a non-standard build process. (May not be a big deal, but could cause things to break in the future.)

Install the Wheel manually (offline mode)

Download the latest "Wheel" file file from PyPI, copy it to the destination server and install with pip.

Offline pip install:

```
pip install ~/Downloads/kintyre-splunk-conf-0.4.2-py2.py3-none-any.whl
```

Install with Splunk's Python

Deprecated since version 0.6.0: Don't do this anymore. Please use the KSCONF App for Splunk instead.

Splunk Enterprise 6.x and later installs an embedded Python 2.7 environment. However, Splunk does not provide packing tools (such as `pip` or the `distutils` standard library which is required to bootstrap install `pip`). For these reasons, it's typically easier and cleaner to install `ksconf` with the system provided Python. However, sometimes the system-provided Python environment is the wrong version, is missing (like on Windows), or security restrictions prevent the installation of additional packages. In such cases, Splunk's embedded Python becomes a beacon of hope.

On Linux or Mac

Download the latest "Wheel" file file from PyPI. The path to this download will be set in the `pkg` variable as shown below.

Setup the shell:

```
export SPLUNK_HOME=/opt/splunk
export pkg=~/Downloads/kintyre_splunk_conf-0.4.9-py2.py3-none-any.whl
```

Run the following:

```
cd $SPLUNK_HOME
mkdir Kintyre
cd Kintyre
# Unzip the 'kconf' folder into SPLUNK_HOME/Kintyre
unzip "$pkg"

cat > $SPLUNK_HOME/bin/ksconf <<HERE
#!/bin/sh
export PYTHONPATH=$PYTHONPATH:$SPLUNK_HOME/Kintyre
exec $SPLUNK_HOME/bin/python -m ksconf \$*
HERE
chmod +x $SPLUNK_HOME/bin/ksconf
```

Test the install:

```
ksconf --version
```

On Windows

1. Open a browser and download the latest "Wheel" file file from PyPI.
2. Rename the `.whl` extension to `.zip`. (This may require showing file extensions in Explorer.)
3. Extract the zip file to a temporary folder. (This should create a folder named "ksconf")
4. Create a new folder called "Kintyre" under the Splunk installation path (aka `SPLUNK_HOME`) By default this is `C:\Program Files\Splunk`.
5. Copy the "ksconf" folder to `%SPLUNK_HOME%\Kintyre`.
6. Create a new batch file called `ksconf.bat` and paste in the following. Be sure to adjust for a non-standard `%SPLUNK_HOME%` value, if necessary.

   ```
   @echo off
   SET SPLUNK_HOME=C:\Program Files\Splunk
   ```

(continues on next page)

(continued from previous page)

```
SET PYTHONPATH=%SPLUNK_HOME%\bin;%SPLUNK_HOME%\Python-2.7\Lib\site-packages\win32;
↪%SPLUNK_HOME%\Python-2.7\Lib\site-packages;%SPLUNK_HOME%\Python-2.7\Lib
SET PYTHONPATH=%PYTHONPATH%;%SPLUNK_HOME%\Kintyre
CALL "%SPLUNK_HOME%\bin\python.exe" -m ksconf %*
```

7. Move `ksconf.bat` to the `Splunk\bin` folder. (This assumes that `%SPLUNK_HOME%/bin` is part of your `%PATH%`. If not, add it, or find an appropriate install location.)
8. Test this by running `ksconf --version` from the command line.

3.12.4 Offline installation

Installing ksconf to an offline or network restricted computer requires three steps: (1) download the latest packages from the Internet to a staging location, (2) transfer the staged content (often as a zip file) to the restricted host, and (3) use pip to install packages from the staged copy. Fortunately, pip makes offline workflows quite easy to achieve. Pip can download a python package with all dependencies stored as wheels files into a single directory, and pip can be told to install from that directory instead of attempting to talk to the Internet.

The process of transferring these files is very organization-specific. The example below shows the creation of a tarball (since `tar` is universally available on Unix systems), but any acceptable method is fine. If security is a high concern, this step is frequently where safety checks are implemented. For example, antivirus scans, static code analysis, manual inspection, and/or comparison of cryptographic file hashes.

One additional use-case for this workflow is to ensure the exact same version of all packages are deployed consistently across all servers and environments. Often building a `requirements.txt` file with `pip freeze` is a more appropriate solution. Or consider using `pipenv lock` for even more security benefits.

Offline installation steps

Important: Pip must be installed on the destination server for this process to work. If pip is NOT installed see the *Offline installation of pip* section below.

Step 1: Use pip to download the latest package and their dependencies. Be sure to use the same version of python that is running on destination machine

```
# download packages
python2.7 -m pip download -d ksconf-packages kintyre-splunk-conf
```

A new directory named 'ksconf-packages' will be created and will contain the necessary `*.whl` files.

Step 2: Transfer the directory or archive to the remote computer. Insert whatever security and file copy procedures necessary for your organization.

```
# Compress directory (on staging computer)
tar -czvf ksconf-packages.tgz ksconf-packages

# Copy file using whatever means
scp ksconf-packages.tgz user@server:/tmp/ksconf-packages.tgz

# Extract the archive (on destination server)
tar -xzvf ksconf-packages.tgz
```

Step 3:

```
# Install ksconf package with pip
pip install --no-index --find-links=ksconf-packages kntyre-splunk-conf

# Test the installation
ksconf --version
```

The `ksconf-packages` folder can now safely be removed.

Offline installation of pip

Use the recommended `pip` install procedures listed elsewhere if possible. But if a remote bootstrap of pip is your only option, then here are the steps. (This process mirrors the steps above and can be combined, if needed.)

Step 1: Fetch bootstrap script and necessary wheels

```
mkdir ksconf-packages
curl https://bootstrap.pypa.io/get-pip.py -o ksconf-packages/get-pip.py
python2.7 -m pip download -d /tmp/my_packages pip setuptools wheel
```

The `ksconf-pacakges` folder should contain 1 script, and 3 wheel (`*.whl`) files.

Step 2: Archive and/or copy to offline server

Step 3: Bootstrap pip

```
sudo python get-pip.py --no-index --find-links=ksconf-packages/

# Test with
pip --version
```

Use pip without installing it

If you have a copy of the `pip*.whl` (wheel) file, then it can be executed directly by python. This can be used to run `pip` without actually installing it, or for install pip initially (bypassing the `get-pip.py` script step noted above.)

Here's an example of how this could work:

Step 1: Download the pip wheel on a machine where `pip` works, by running:

```
pip download pip -d .
```

This will create a file like `pip-19.0.1-py2.py3-none-any.whl` in the current working directory.

Step 2: Copy the pip wheel to another machine (likely where pip isn't installed.)

Step 3: Execute the wheel by running:

```
python pip-19.0.1-py2.py3-none-any.whl/pip list
```

Just substitute the `list` command with whatever action you need (like `install` or whatever)

3.12.5 Frequent gotchas

PIP Install TLS Error

If `pip` throws an error message like the following:

```
There was a problem confirming the ssl certificate: [SSL: TLSV1_ALERT_PROTOCOL_VERSION]␣
↪tlsv1 alert protocol version
...
No matching distribution found for setuptools
```

The problem is likely caused by changes to PyPI website in April 2018 when support for TLS v1.0 and 1.1 were removed. Downloading new packages requires upgrading to a new version of pip. Like so:

Upgrade pip as follows:

```
curl https://bootstrap.pypa.io/get-pip.py | python
```

Note: Use `sudo python` above if not in a virtual environment.

Helpful links:

- Not able to install Python packages [SSL: TLSV1_ALERT_PROTOCOL_VERSION]
- 'pip install' fails for every package ("Could not find a version that satisfies the requirement")

No module named 'command.install'

If, while trying to install `pip` or run a `pip` command you see the following error:

```
ImportError: No module named command.install
```

Likely this is because you are using a crippled version of Python; like the one that ships with Splunk. This won't work. Either get a pre-package version (the `.pyz` file or install using the OS-level Python.

3.12.6 Troubleshooting

Here are a few fact gathering type commands that may help you begin to track down problems.

Check Python version

Check your installed python version by running:

```
python --version
```

Note that Linux distributions and Mac OS X that ship with multiple version of Python may have renamed this to `python2`, `python2.7` or similar.

Check PIP Version

```
pip --version
```

If you are running a different python interpreter version, you can instead run this as:

```
python2.7 -m pip --version
```

Validate the install

Confirm installation with the following command:

```
ksconf --version
```

If this works, it means that `ksconf` installed and is part of your `PATH` and should be useable everywhere in your system. Go forth and conquer!

If this doesn't work here are a few things to try:

1. Check that your `PATH` is set correctly.
2. Try running ksconf as a "module" (sometimes works around a PATH issue). Run `python -m ksconf`
3. If you're running the Splunk app, try running the following:

   ```
   cd $SPLUNK_HOME/etc/apps/ksconf/bin/lib
   python -m ksconf --version
   ```

   If this works, then the issue has something to do with your path.

It may be helpful to uninstall (remove) the Splunk app and reinstall from scratch.

3.12.7 Resources

- Python packaging docs provide a general overview on installing Python packages, how to install per-user vs install system-wide.
- Install PIP docs explain how to bootstrap or upgrade pip the Python packaging tool. Recent versions of Python come with this by default, but releases before Python 2.7.9 do not.

3.13 License

(continues on next page)

(continued from previous page)

```
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as "Not a Contribution."

      "Contributor" shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
      subsequently incorporated within the Work.

   2. Grant of Copyright License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribute the
      Work and such Derivative Works in Source or Object form.

   3. Grant of Patent License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      (except as stated in this section) patent license to make, have made,
      use, offer to sell, sell, import, and otherwise transfer the Work,
      where such license applies only to those patent claims licensable
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contribution(s)
      with the Work to which such Contribution(s) was submitted. If You
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the Work
      or a Contribution incorporated within the Work constitutes direct
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall terminate
      as of the date such litigation is filed.

   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:
```

(continues on next page)

(continued from previous page)

```
      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
          notices within Derivative Works that You distribute, alongside
          or as an addendum to the NOTICE text from the Work, provided
          that such additional attribution notices cannot be construed
          as modifying the License.

      You may add Your own copyright statement to Your modifications and
      may provide additional or different license terms and conditions
      for use, reproduction, or distribution of Your modifications, or
      for any such Derivative Works as a whole, provided Your use,
      reproduction, and distribution of the Work otherwise complies with
      the conditions stated in this License.

   5. Submission of Contributions. Unless You explicitly state otherwise,
      any Contribution intentionally submitted for inclusion in the Work
      by You to the Licensor shall be under the terms and conditions of
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede or modify
      the terms of any separate license agreement you may have executed
      with Licensor regarding such Contributions.

   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
      origin of the Work and reproducing the content of the NOTICE file.

   7. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
```

(continues on next page)

(continued from previous page)

3.14 API Reference

3.14.1 ksconf

Subpackages

ksconf.commands package

Module contents

class `ksconf.commands.`**`KsconfCmd`**(*name*)

Bases: `object`

Ksconf command specification base class.

`add_parser`(*subparser*)

`description = None`

`format = 'default'`

`help = None`

`launch`(*args*)

Handle flow control between pre_run() / run() / post_run()

`maturity = 'alpha'`

`post_run`(*args*, *exec_info=None*)

Any custom clean up work that needs done. Always called if run() was. Presence of exc_info indicates failure.

`pre_run`(*args*)

Pre-run hook. Any exceptions here prevent run() from being called.

`register_args`(*parser*)

This function in passed the

`run`(*args*)

Actual works happens here. Return code should be an EXIT_CODE_* from consts.

class `ksconf.commands.`**`ConfDirProxy`**(*name*, *mode*, *parse_profile=None*)

Bases: `object`

`get_file`(*relpath*)

class `ksconf.commands.`**`ConfFileProxy`**(*name*, *mode*, *stream=None*, *parse_profile=None*, *is_file=None*)

Bases: `object`

`close`()

`data`

`dump`(*data*)

is_file()

load(*profile=None*)

reset()

set_parser_option(***kwargs*)
Setting a key to None will remove that setting.

stream

unlink()

class ksconf.commands.**ConfFileType**(*mode='r'*, *action='open'*, *parse_profile=None*, *accept_dir=False*)
Bases: object

Factory for creating conf file object types; returns a lazy-loader ConfFile proxy class

Started from argparse.FileType() and then changed everything. With our use case, it's often necessary to delay writing, or read before writing to a conf file (depending on weather or not –dry-run mode is enabled, for example.)

Instances of FileType are typically passed as type= arguments to the ArgumentParser add_argument() method.

Parameters

- **mode** (str) – How the file is to be opened. Accepts "r", "w", and "r+".
- **action** (str) – Determine how much work should be handled during argument parsing vs handed off to the caller. Supports 'none', 'open', 'load'. Full descriptions below.
- **parse_profile** – parsing configuration settings passed along to the parser
- **accept_dir** (bool) – Should the CLI accept a directory of config files instead of an individual file. Defaults to *False*.

Values for action

Action	Description
none	No preparation or testing is done on the filename.
open	Ensure the file exists an can be opened.
load	Ensure the file can be opened and parsed successfully.

Once invoked, instances of this class will return a ConfFileProxy object, or a ConfDirProxy object if a directory is passed in via the CLI.

ksconf.commands.**dedent**(*text*)
Remove any common leading whitespace from every line in *text*.

This can be used to make triple-quoted strings line up with the left edge of the display, while still presenting them in the source code in indented form.

Note that tabs and spaces are both treated as whitespace, but they are not equal: the lines " hello" and "thello" are considered to have no common leading whitespace. (This behaviour is new in Python 2.5; older versions of this module incorrectly expanded tabs before searching for common leading whitespace.)

`ksconf.commands.`**`get_all_ksconf_cmds`**(*ignore_errors=False*)

`ksconf.commands.`**`get_entrypoints`**

ksconf.conf package

Submodules

ksconf.conf.delta module

class `ksconf.conf.delta.`**`DiffGlobal`**(*type*)

Bases: `tuple`

type

Alias for field number 0

class `ksconf.conf.delta.`**`DiffOp`**(*tag*, *location*, *a*, *b*)

Bases: `tuple`

a

Alias for field number 2

b

Alias for field number 3

location

Alias for field number 1

tag

Alias for field number 0

class `ksconf.conf.delta.`**`DiffStanza`**(*type*, *stanza*)

Bases: `tuple`

stanza

Alias for field number 1

type

Alias for field number 0

class `ksconf.conf.delta.`**`DiffStzKey`**(*type*, *stanza*, *key*)

Bases: `tuple`

key

Alias for field number 2

stanza

Alias for field number 1

type
Alias for field number 0

`ksconf.conf.delta.`**`compare_cfgs`**(*a*, *b*, *allow_level0=True*)
Return list of 5-tuples describing how to turn a into b.

Note: The *Opcode* tags borrowed from `SequenceMatcher` class in the `difflib` standard Python module.

Each tuple takes the form:

(tag, location, a, b)

tag:

Value	Meaning
'replace'	same element in both, but different values.
'delete'	remove value b
'insert'	insert value a
'equal'	same values in both

location is a tuple that can take the following forms:

Tuple form	Description
(0)	Global file level context (e.g., both files are the same)
(1, stanza)	Stanzas are the same, or completely different (no shared keys)
(2, stanza, key)	Key level, indicating

Possible alternatives:

https://dictdiffer.readthedocs.io/en/latest/#dictdiffer.patch

`ksconf.conf.delta.`**`show_diff`**(*stream*, *diffs*, *headers=None*)

`ksconf.conf.delta.`**`show_text_diff`**(*stream*, *a*, *b*)

`ksconf.conf.delta.`**`summarize_cfg_diffs`**(*delta*, *stream*)
Summarize a delta into a human readable format. The input *delta* is in the format produced by the compare_cfgs() function.

ksconf.conf.merge module

`ksconf.conf.merge.`**`merge_conf_dicts`**(**dicts*)

`ksconf.conf.merge.`**`merge_conf_files`**(*dest*, *configs*, *dry_run=False*, *banner_comment=None*)

ksconf.conf.parser module

Parse and write Splunk's .conf files

According to this doc:

https://docs.splunk.com/Documentation/Splunk/7.2.3/Admin/Howtoeditaconfigurationfile

1. Comments must start at the beginning of a line (#)
2. Comments may not be after a stanza name or on an attribute's value
3. Supporting encoding is UTF-8 (and therefore ASCII too)

exception `ksconf.conf.parser.`**`ConfParserException`**

Bases: `Exception`

exception `ksconf.conf.parser.`**`DuplicateKeyException`**

Bases: `ksconf.conf.parser.ConfParserException`

exception `ksconf.conf.parser.`**`DuplicateStanzaException`**

Bases: `ksconf.conf.parser.ConfParserException`

class `ksconf.conf.parser.`**`Token`**

Bases: `object`

Immutable token object. deepcopy returns the same object

`ksconf.conf.parser.`**`cont_handler`**(*iterable*, *continue_re=re.compile('^(.*)\\\\\\$')*, *breaker='\n'*)

Look for trailing backslashes ("\") which indicate a value for an attribute is split across multiple lines. This function will group such lines together, and pass all other lines through as-is. Note that the continuation character must be the very last character on the line, trailing whitespace is not allowed.

Parameters

- **`iterable`** (`iter`) – lines from a configuration file
- **`continue_re`** – regular expression to detect the continuation character
- **`breaker`** – joining string when combining continued lines into a single string. Default '\n'

Returns lines of text

Return type str

`ksconf.conf.parser.`**`detect_by_bom`**(*path*)

`ksconf.conf.parser.`**`inject_section_comments`**(*section*, *prepend=None*, *append=None*)

`ksconf.conf.parser.`**`parse_conf`**(*stream*, *profile={'dup_key': 'overwrite', 'dup_stanza': 'exception', 'keep_comments': True, 'strict': True}*, *encoding=None*)

Parse a .conf file. This is a wrapper around `parse_conf_stream()` that allows filenames or stream to be passed in.

Parameters

- **stream** (str, file) – the path to a configuration file or open file-like object to be parsed
- **profile** – parsing configuration settings
- **encoding** – Defaults to the system default, "uft-8"

Returns a mapping of the stanza and attributes. The resulting output is accessible as [stanaza][attribute] -> value

Return type dict

`ksconf.conf.parser.`**parse_conf_stream**(*stream*, *keys_lower=False*, *handle_conts=True*, *keep_comments=False*, *dup_stanza='exception'*, *dup_key='overwrite'*, *strict=False*)

`ksconf.conf.parser.`**section_reader**(*stream*, *section_re=re.compile('^[\\s\\t]*\\[(.*)\\]\\s*$')*)

This generator break a configuration file stream into sections. Each section contains a name and a list of text lines held within that section.

Sections that have no entries may be dropped. Any lines before the first section are send back with the section name of None.

Parameters

- **stream** (file) – configuration file input stream
- **section_re** – regular expression for detecting stanza headers

Returns sections in the form of *(section_name, lines_of_text)*

Return type tuple

`ksconf.conf.parser.`**smart_write_conf**(*filename*, *conf*, *stanza_delim='\n'*, *sort=True*, *temp_suffix='.tmp'*)

`ksconf.conf.parser.`**splitup_kvpairs**(*lines*, *comments_re=re.compile('^\\s*[#;]')*, *keep_comments=False*, *strict=False*)

Break up 'attribute=value' entries in a configuration file.

Parameters

- **lines** (iter) – the body of a stanza containing associated attributes and values
- **comments_re** – Regular expression used to detect comments.
- **keep_comments** (bool, optional) – Should comments be preserved in the output. Defaults to *False*.
- **strict** (bool, optional) – Should unknown content in the stanza stop processing. Defaults to *False* allowing "junk" to be silently ignored allowing for a best-effort parse.

Returns iterable of (attribute,value) tuples

`ksconf.conf.parser.`**write_conf**(*stream*, *conf*, *stanza_delim='\n'*, *sort=True*) `ksconf.conf.parser.`**`write_conf_stream`**(*stream*, *conf*, *stanza_delim='\n'*, *sort=True*)

Module contents

ksconf.util package

Submodules

ksconf.util.compare module

`ksconf.util.compare.`**`file_compare`**(*fn1*, *fn2*)

`ksconf.util.compare.`**`fileobj_compare`**(*f1*, *f2*)

ksconf.util.completers module

`ksconf.util.completers.`**`DirectoriesCompleter`**(**args*, ***kwargs*)

`ksconf.util.completers.`**`FilesCompleter`**(**args*, ***kwargs*)

`ksconf.util.completers.`**`autocomplete`**(**args*, ***kwargs*)

ksconf.util.file module

`class` `ksconf.util.file.`**`ReluctantWriter`**(*path*, **args*, ***kwargs*)

Bases: `object`

Context manager to intelligently handle updates to an existing file. New content is written to a temp file, and then compared to the current file's content. The file file will be overwritten only if the contents changed.

`ksconf.util.file.`**`dir_exists`**(*directory*)

Ensure that the directory exists

`ksconf.util.file.`**`file_fingerprint`**(*path*, *compare_to=None*)

`ksconf.util.file.`**`file_hash`**(*path*, *algorithm='sha256'*)

`ksconf.util.file.`**`match_bwlist`**(*value*, *bwlist*, *escape=True*)

`ksconf.util.file.`**`relwalk`**(*top*, *topdown=True*, *onerror=None*, *followlinks=False*)

Relative path walker Like os.walk() except that it doesn't include the "top" prefix in the resulting 'dirpath'.

`ksconf.util.file.`**`smart_copy`**(*src*, *dest*)

Copy (overwrite) file only if the contents have changed.

ksconf.util.terminal module

`ksconf.util.terminal.`**`tty_color`**(*stream*, **codes*)

Module contents

`ksconf.util.`**`debug_traceback`**()

If the 'KSCONF_DEBUG' environmental variable is set, then show a stack trace.

ksconf.vc package

Submodules

ksconf.vc.git module

`class` `ksconf.vc.git.`**`GitCmdOutput`**(*cmd*, *returncode*, *stdout*, *stderr*, *lines*)

Bases: `tuple`

`cmd`

Alias for field number 0

`lines`

Alias for field number 4

`returncode`

Alias for field number 1

`stderr`

Alias for field number 3

`stdout`

Alias for field number 2

`ksconf.vc.git.`**`git_cmd`**(*args*, *shell=False*, *cwd=None*, *capture_std=True*, *encoding='utf-8'*)

`ksconf.vc.git.`**`git_cmd_iterable`**(*args*, *iterable*, *cwd=None*, *cmd_len=1024*)

`ksconf.vc.git.`**`git_is_clean`**(*path=None*, *check_untracked=True*, *check_ignored=False*)

`ksconf.vc.git.`**`git_is_working_tree`**(*path=None*)

`ksconf.vc.git.`**`git_ls_files`**(*path*, **modifiers*)

`ksconf.vc.git.`**`git_status_summary`**(*path*)

`ksconf.vc.git.`**`git_status_ui`**(*path*, **args*)

Module contents

Submodules

ksconf.archive module

`ksconf.archive.`**`GenArchFile`**
    alias of `ksconf.archive.GenericArchiveEntry`

`ksconf.archive.`**`extract_archive`**`(`*archive_name*, *extract_filter=None*`)`

`ksconf.archive.`**`gaf_filter_name_like`**`(`*pattern*`)`

`ksconf.archive.`**`gen_arch_file_remapper`**`(`*iterable*, *mapping*`)`

`ksconf.archive.`**`sanity_checker`**`(`*interable*`)`

ksconf.consts module

ksconf.setup_entrypoints module

Defines all command prompt entry points for CLI actions

This is a silly hack that serves 2 purposes:

1. It works around an apparent Python 3.4/3.5 bug on Windows where [options.entry_point] in setup.cfg is ignored hence 'ksconf' isn't installed as a console script and custom ksconf_* entry points are not available. (So no CLI commands are available)
2. **It allows for fallback mechanism when**
    (a) running unit tests (can happen before install)
    (b) if entrypoints or pkg_resources are not available at run time (Splunk's embedded python)

`class` `ksconf.setup_entrypoints.`**`Ep`**`(`*name*, *module_name*, *object_name*`)`
    Bases: `tuple`

    `module_name`
        Alias for field number 1

    `name`
        Alias for field number 0

    `object_name`
        Alias for field number 2

`class` `ksconf.setup_entrypoints.`**`LocalEntryPoint`**`(`*data*`)`
    Bases: `object`

    Bare minimum stand-in for entrypoints.EntryPoint

    `load``()`

`ksconf.setup_entrypoints.`**`get_entrypoints_fallback`**`(`*group*`)`

`ksconf.setup_entrypoints.`**`get_entrypoints_setup`**`()`

Module contents

ksconf - Kintyre Splunk CONFig tool

Design goals:

- Multi-purpose go-to `.conf` tool.
- Dependability
- Simplicity
- No eternal dependencies (single source file, if possible; or packable as single file.)
- Stable CLI
- Good scripting interface for deployment scripts and/or git hooks

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Bibliography

[SPLKDOC1] https://docs.splunk.com/Documentation/Splunk/7.2.3/Admin/Configurationfiledirectories

Python Module Index

Index