NAME

caspar - Makefile snippets for common tasks

SYNOPSIS

In a Makefile, write either

```
include caspar/mk/caspar.mk
```

or

```
include caspar/mk/docbook.mk
```

or

```
include caspar/mk/pod.mk
```

.

DESCRIPTION

Caspar offers Makefile snippets for common tasks, like installing (configuration) files, or typesetting LaTeX, DocBook XML and DocBook SGML documents.

The typesetting functionality is delivered by *docbook.mk* and *pod.mk*. This is documented in **caspar-typesetting(7)**.

The installing-stuff functionality is delivered by *caspar.mk*. (That's what the rest of the manual will talk about.) It enables one to run 'make install' from within a tree which typically holds configuration files, managed using Subversion (or git or any other version control system, for that matter).

It is useful in cases like this: all configuration files of some host are under version control, and, after commiting a change to CVS, you want to be able to easily install the new configuration file on the host.

With **caspar**, all you have to do is specify the hostname in one place, and specify the name of the target directory in each CVS directory.

It is comparable with other tools for Unix system administrators like puppet and cfengine. Main difference: the caspar code consists of less than 100 lines of GNU Make.

USAGE

Within a CVS tree, create a file *include/install.mk*, with contents like e.g.

```
csp_UHOST = root@some.host.somewhere
include caspar/mk/caspar.mk
```

. Within each other directory of the CVS tree which holds files, create a Makefile, which looks like e.g.

```
csp_DIR = /some/dir/ectory/
include ../../include/install.mk
```

. If you'd like to use the *install-recursive* target too, in directories which hold subdirectories (but not files), you'll have to create a Makefile which looks something like

```
include ../../include/install.mk
```

. From within the CVS tree, one can call:

```
make <filename>-diff
make <filename>-install
make diff
make install
make load
make
make install-recursive
```

Calling `make <filename>-diff` shows the diff between the local file and the file as currently installed on the remote location. Calling `make install` (or `make`) now will scp all files in the current directory to the remote location. The *install-recursive* target descends down the tree, and calls `make install` in each subdirectory.

Of course, you'll have to be able to ssh directly as root to the remote host to get this working (if you'd like to use `csp_UHOST = root@some.host.somewhere`). If you don't like this, and would like to have a `PermitRootLogin no` in your `/etc/ssh/sshd_config`, you can use csp_sucp(1). See below.

VARIABLES

The variables one can set in a calling Makefile are:

`csp_UHOST`
: user@host, reachable using `$(csp_PUSH)` (which is scp by default)

`csp_UHOSTS`
: space separated list of user@host items, reachable using `$(csp_PUSH)`

`csp_DIR`
: directory on host, reachable using function `$(csp_PUSH)`

`csp_PUSH`
: make function for pushing files to remote location. Usually, this is a wrapper around a script or program. The function will be passed 4 arguments: *[user@]host*, remote *directory* and local *filename*. *[user@]host* will be set to all elements of `$(csp_UHOSTS)`; *directory* will be set to `$(csp_DIR)`. Currently, `$(csp_scp_FUNC)`, `$(csp_cp_FUNC)` and `$(csp_sucp_FUNC)` are supported as push plugins. If `csp_PUSH` is unset, the default `$(csp_scp_FUNC)` is used.

`csp_LOAD`
: the 'load' target depends upon these targets.

`csp_BUILD`
: the 'build' target depends upon these targets.

`csp_CP`
: cp binary, just "cp" by default

`csp_SCP`
: scp binary, just "scp" by default

`csp_SUCP`

script wrapping sudo in ssh, "csp_sucp" by default

`csp_EXTRAFILES`
: extra files which should be installed. Can be used to include files starting with a dot.

`csp_TABOOFILES`
: files which should never be installed. Set to `Makefile .%.swp %~ #%# pod2htmd.tmp pod2htmi.tmp` by default.

`csp_TABOOFILES_ADD`
: extra files which should never be installed; added to list in `csp_TABOOFILES`.

`csp_TABOOFILES_SKIP`
: files which should be installed, even if in initial `csp_TABOOFILES` list. Removed from `csp_TABOOFILES` list.

`csp_TABOODIRS`
: directories to exclude in install-recursive target. set to `CVS .svn` by default.

`csp_TABOODIRS_ADD`, `csp_TABOODIRS_SKIP`
: see `csp_TABOOFILES` equivalents.

`csp_UHOSTS_SUBSET`
: override csp_UHOSTS: don't push to csp_UHOSTS, but to the intersection of this space separated list of user@host items and csp_UHOSTS.

The following variables might get phased out or removed soonish:

`csp_CPFLAGS`
: extra arguments to pass to cp invocation, none by default

`csp_SCPFLAGS`
: extra arguments to pass to scp invocation, e.g. `'-i .ssh/id_rsa-root'`

EXAMPLES

Some examples:

Using csp_UHOST

This is the simplest way to use caspar. *Makefile* is

```
csp_UHOST = root@some.host.somewhere
csp_DIR = /etc/
include caspar/mk/caspar.mk
```

Now, running "make" will scp all files in the current directory to `root@some.host.somewhere:/etc/`.

More hosts, not scp but sudo via ssh: using csp_PUSH

Makefile is

```
csp_UHOSTS = root@some.host.somewhere root@some.other.host
csp_PUSH = $(csp_sucp_FUNC)
csp_DIR = /etc/
include caspar/mk/caspar.mk
```

Now, running "make" will use csp_sucp(1) to install all files in the current directory to both root@some.host.somewhere:/etc/ and root@some.other.host:/etc/. If a file named `fstab` is present in the current directory, running "make fstab-install" will just install that one file. If you need to sudo(1) to another user on the remote host, add something like

```
csp_XARG = postgres
```

. (If such a username is not supplied, sudo (and csp_sucp) use the default: root.)

Overriding csp_UHOSTS: csp_UHOSTS_SKIP

If one or some of your hosts are temporarily unavailable, and you'd like to push your files to the hosts which are alive, you can temporarily override your csp_UHOSTS. E.g., when some.other.host is not available:

```
% cat Makefile
csp_UHOSTS = root@some.host.somewhere root@some.other.host
csp_DIR = /etc/
include caspar/mk/caspar.mk

% make install csp_UHOSTS_SKIP=root@some.other.host
scp hosts root@some.host.somewhere:/etc/
scp fstab root@some.host.somewhere:/etc/
```

Overriding csp_UHOSTS in a smart way: csp_UHOSTS_SUBSET. Using multiple groups of hosts. Recursive make made easy.

If you have lots of subdirectories holding information for lots of groups of hosts, while this run you just want to install for a small group (or 1) hosts, csp_UHOSTS_SUBSET is useful. Suppose your casparized tree looks like

```
Makefile

apache/include/install.mk
apache/etc/apache2/Makefile
apache/etc/apache2/envvars

php/include/install.mk
php/etc/php4/apache/Makefile
php/etc/php4/apache/php.ini

grub/include/install.mk
grub/boot/grub/Makefile
grub/boot/grub/menu.lst
logrotate/include/install.mk
logrotate/etc/Makefile
logrotate/etc/logrotate.conf

nrpe/include/install.mk
nrpe/debian/etc/default/Makefile
nrpe/debian/etc/default/nagios-nrpe-server
```

The file *apache/etc/apache2/Makefile* is:

```
csp_DIR = /etc/apache2/
include ../../include/install.mk
```

(all other *Makefile*s are similar). The file *apache/include/install.mk* is

```
csp_UHOSTS = root@a root@b
include caspar/mk/caspar.mk
```

The file *php/include/install.mk* is the same. The files *grub/include/install.mk* and *logrotate/include/install.mk* are

```
csp_UHOSTS = root@d root@e root@f root@g
include caspar/mk/caspar.mk
```

The file *nrpe/include/install.mk* is

```
csp_UHOSTS = root@d root@e root@f root@n
include caspar/mk/caspar.mk
```

The toplevel *Makefile* is

```
dirs = $(patsubst %/Makefile,%,$(shell find * -mindepth 1
 -name Makefile))

all:
    for i in $(dirs); do $(MAKE) -$(MAKEFLAGS) -C $$i; done

install
    for i in $(dirs); do $(MAKE) -$(MAKEFLAGS) -C $$i install; done

load
    for i in $(dirs); do $(MAKE) -$(MAKEFLAGS) -C $$i load; done
```

(we don't feel like sticking a *Makefile* in all non-leaf nodes of our tree).

Now, when running "`csp_UHOSTS_SUBSET='root@e root@f root@m root@n' make`" in the toplevel, caspar just takes the intersection of csp_UHOSTS_SUBSET and csp_UHOSTS for each csp_UHOSTS list. So, caspar will not push anything for *apache/* and *php/*. The files *grub/boot/grub/menu.lst* and *logrotate/etc/logrotate.conf* will get pushed to `root@e` and `root@f` only. The file *nrpe/debian/etc/default/nagios-nrpe-server* will get pushed to `root@e`, `root@f` and `root@n`.

This is often better than just overriding csp_UHOSTS on the commandline (or in your shell's environment): if the intersection of the original csp_UHOSTS and your new csp_UHOSTS is empty, chances are big you've just forgotten to clean your environment.

Creating remote directories if needed

Makefile is

```
csp_DIR = /some/dir/ectory/
csp_PUSH = $(csp_scpmkdir_FUNC)
csp_UHOST = root@some.host.somewhere
include caspar/mk/caspar.mk
```

Now, before calling scp, caspar will run 'mkdir -p' to create any missing remote directories.

Using csp_CP and csp_LOAD
username/etc/Makefile is

```
csp_UHOST = dummy
csp_PUSH = $(csp_cp_FUNC)
csp_DIR = $(HOME)/etc/
csp_LOAD = crontab-load
include ../include/install.mk

crontab-load = crontab $(csp_DIR)/crontab
```

while *../include/install.mk* is just

```
include caspar/mk/caspar.mk
```

. Setting `csp_PUSH` to `$(csp_cp_FUNC)` causes cp(1) to get executed by "make install" (not scp(1)). Setting `csp_LOAD` causes "make load" to execute the crontab command. Just running "make" is OK too, since "make" calls both "make install" and "make load".

Using csp_DIR, csp_LOAD and install(1)
To install a file on the local host, create e.g. a file *etc/uruk/Makefile* like:

```
csp_UHOST = dummy
csp_DIR = /etc/uruk/
csp_PUSH = $(csp_install_FUNC)
csp_LOAD = uruk-load

include caspar/mk/caspar.mk

uruk-load = sudo invoke-rc.d uruk force-reload
```

Using csp_DIR and csp_LOAD, take 2
etc/Makefile is

```
csp_DIR = /etc/
csp_LOAD = aliases-load
include ../include/install.mk

aliases-load = $(csp_SSH) $1 "cd /etc && postalias aliases; postfix reload"
```

while *../include/install.mk* is

```
csp_UHOST = root@some.host.somewhere
include caspar/mk/caspar.mk
```

Building files locally
Note: csp_BUILD is deprecated. You should not use it. If you'd like to build some files locally from local sources, before installing the just built files, do e.g.:

```
csp_UHOST = root@some.host.somewhere
csp_DIR = /etc/
csp_EXTRAFILES = sshd_config
csp_TABOOFILES_ADD = sshd_config.m4

include caspar/mk/caspar.mk

sshd_config: sshd_config.m4
    m4 $< > $@
```

List all source files in csp_TABOOFILES_ADD: this way, they won't get installed on the csp_UHOST.

Using csp_sucp_FUNC and csp_LOAD
If you'd like to use csp_sucp and want a 'load' target, do something like:

```
csp_PUSH = $(csp_sucp_FUNC)
csp_UHOST = foobar.example.com
csp_DIR = /etc/uruk/
csp_LOAD = rc-load

include caspar/mk/caspar.mk

rc-load = $(csp_SSH) $1 sudo invoke-rc.d uruk force-reload
```

Using the "verify" target
If you want to do some syntax check on the local host for a configuration file, define a -verify target (or pattern target):

```
bar.pem-verify: bar.pem.asc bar.pem
     gpg --verify $^

%.php-verify: %.php
    php -l $<
```

This command will be invoked when you "make verify" and "make install".

Using the "check" target
If you want to do some syntax check on the remote host, before loading the just installed configuration file, do

```
csp_UHOST = foobar.example.com
csp_DIR = /etc/
csp_LOAD = my-load
csp_CHECK = my-check

my-check = $(csp_SSH) $1 do-check-stuff
my-load = $(csp_SSH) $1 do-load-stuff

include caspar/mk/caspar.mk
```

This way, "make load" won't cause the file to load if the check fails (which is probably what you want). You could even give the file a temporary name and have the load target rename it to its real name only after a successful check. As a convenience, if a foo-check variable is defined, it is automatically added as a dependency for any csp_LOAD targets named foo or foo-something.

Combining the csp_LOAD target with multiple hosts; building files remotely

Here's an example doing some preprocessing on the remote hosts before reloading the configuration:

```
csp_DIR = /etc/ssh/
csp_UHOSTS = root@some.host.somewhere root@some.other.host
csp_LOAD = sshd_config-load
sshd_config-load = $(csp_SSH) $1 "set -e; \
       cd $(csp_DIR); \
       m4 sshd_config.m4 >sshd_config; \
       service ssh reload"
include caspar/mk/caspar.mk
```

(The old, deprecated way is to explicitly specify the loop over the hosts:

```
csp_DIR = /etc/ssh/
sshd_config-load:
    for suh in $(csp_UHOSTS); do \
      $(csp_SSH) $$suh "set -e;
       cd $(csp_DIR); \
       m4 sshd_config.m4 > sshd_config; \
       service ssh reload"; \
    done
```

. You should not use this method. Please consider rewriting any existing caspar snippets using the simpler method described above.)

Using the csp_TABOOFILES_{ADD,SKIP} variables; another way to perform remote builds

Using the csp_TABOOFILES_{ADD,SKIP} variables is handy if you want to *install* a *Makefile*, instead of using it: Create *Makefile* just as you'd like to have it installed on the remote location. Now, create *GNUmakefile* as e.g.

```
csp_TABOOFILES_SKIP = Makefile
csp_TABOOFILES_ADD = GNUmakefile

csp_DIR = /etc/foobar/
csp_UHOST = root@some.host.somewhere
include caspar/mk/caspar.mk

load:
      $(csp_SSH) $(csp_UHOST) "make -C $(csp_DIR)"
```

Now, `make install` and `make load` will do the right thing.

Using the csp_EXTRAFILES variable

Using the csp_EXTRAFILES variable can be handy if you want to install files with a leading dot. E.g.:

```
csp_EXTRAFILES = .bashrc
csp_UHOST = root@some.host.somewhere
csp_DIR =
include caspar/mk/caspar.mk
```

Overriding csp_UHOSTS
Supply e.g.

```
csp_UHOSTS = root@localhost root@some.host.somewhere
```

in *install.mk*, to install on multiple hosts. Run

```
make filename-install csp_UHOSTS=joe@otherhost
```

to install filename as `joe@otherhost`, instead of the default as given in *install.mk*. If you want to enable passing csp_UHOSTS as a shell environment variable, you'll have to use conditional assignment in your Makefile:

```
csp_UHOSTS ?= root@localhost root@some.host.somewhere
```

This allows it to run

```
% export csp_UHOSTS=foo@bar
% make filename-install
```

to install on `foo@bar`.

Using sudo locally for installing files
If you'd like to install files like

```
sudo cp foo.rc /etc/foobar/
```

you could set up your *Makefile* as

```
csp_DIR = /etc/foobar/
csp_UHOST = dummy
csp_PUSH = sudo cp $(1) $(3)
include caspar/mk/caspar.mk
```

This is like csp_sucp, but without the ssh wrapping: it works on localhost only.

Plugging your own install script in caspar
If your script `foobar` should be called as e.g.

```
foobar --file=fstab --user@host=joe@some.host \
  --dir=/etc/ --debuglevel=3
```

then make sure your *Makefile* features something like

```
csp_foobar_FUNC = foobar --file=$(1) --user@host=$(2) \
  --dir=$(3) --debuglevel=$(4)
csp_PUSH = $(csp_foobar_FUNC)
csp_XARG = 3
```

You can now use `csp_UHOST` and `csp_DIR` just as you're used to.

More advanced tricks
When you don't want to ssh to `root@some.host.somewhere` directly, you could do

```
sudo rsync -az /path/to/your/config_archive /etc
```

on some.host.somewhere (e.g. from cron).

FILES

caspar/mk/caspar.mk, *caspar/mk/docbook.mk*, *caspar/mk/pod.mk*

ENVIRONMENT

For *caspar.mk*: `csp_CP`, `csp_LOAD`, `csp_SCP`, `csp_UHOST`, `csp_PUSH`, ...

BUGS

Very likely, GNU Make is not the best tool for doing the stuff *caspar.mk* is doing. For the list of reported bugs, see http://bugs.debian.org/src:caspar. See also TODO, distributed with the caspar package. (And online at http://mdcc.cx/pub/caspar/caspar-latest/TODO.)

TRIVIA

Caspar is named after Caspar the Friendly Ghost, since that's the title of the Daniel Johnston song I was listening to when deciding to package my homegrown scripts.

AUTHOR

Joost van Baal-IliÄ

SEE ALSO

caspar-typesetting(7) csp_helper(1)

The caspar homepage is at http://mdcc.cx/caspar/ .

The document *"Versiebeheer en software-packages: Waarom en Hoe"* (in Dutch) describes some of the reasons why people might want to use tools like caspar.

Jeroen Hoppenbrouwers blogs about the way he uses caspar, in *"Using Subversion and Caspar to maintain a Linux host"*.

Lots of tools overlap (partly) with caspar in their functionality. Here's a list.

ansible (http://www.ansible.com/); code is on github (https://github.com/ansible/ansible)

cfengine (http://www.cfengine.org/), by Mark Burgess e.a., builds expert systems to administrate and configure large computer networks: it delivers a very big superset of caspar's installation mechanism.

Puppet (http://reductivelabs.com/projects/puppet), also something like a configuration management system.

Here's an older list; a big part of it was collected by Ray Miller (http://users.ox.ac.uk/˜raym/) of Oxford University, and published in the article "Configuration Management with Subversion, YAML and Perl Template Toolkit" in the SANE 2006 (http://www.sane.nl/sane2006/) conference proceedings. FIXME: Check urls, update

docbookmk, by Michael Wiedmann (http://www.miwie.org/docbkmake/) offers probably a superset of Caspar's *docbook.mk* functionality.

latex-make by the LaTeX Utils project on http://gforge.inria.fr/projects/latex-utils/ seems to provide similar functionality as *docbook.mk* for LaTeX documents.

Latexmk by John Collins e.a. on http://www.phys.psu.edu/˜collins/software/latexmk-jcc/ is another implementation of this idea.

SUP, the Software Upgrade Protocol and it's implementation by Carnegie Mellon University offers another way to distribute (configuration)files. Beware though: between Nov 1996 and June 2004, no new release has been published. The Debian (ftp://ftp.debian.org/debian/pool/main/s/sup/) and NetBSD packages are likely still maintained, though.

PIKT (http://www.pikt.org/) is intended primarily for system monitoring, but does do configuration management too.

LCFG (http://www.lcfg.org/) is another configuration management system.

The Arusha Project (ARK, at http://ark.sf.net/) provides a framework for collaborative system administration.

Bcfg2 (http://trac.mcs.anl.gov/projects/bcfg2/) is yet another configuration management system.

quattor (http://quattor.web.cern.ch/) is a system administration toolkit for installation, configuration and management of Unix systems.

rb3 and friends, as written and used by Ray Miller e.a. at Oxford University, (http://users.ox.ac.uk/˜raym/software/configuration-management/).

The *svk* version control system is said to be quite usable for handling configuration file management (without a separate install mechanism like caspar). See also *this discussion on the Debian development list*.

On the http://www.infrastructures.org/ website on automated (Unix) system administration, you can find some thoughts on managing configuration files using a version control system.