General scripts for CLTL

Paul Huygen

26th February 2018

Abstract

this document describes and generates scripts that are useful.

1 Pipeline

1.1 Restore the eSRL server

It turns out that the server for ‘eSRL‘ module may stop to work as advertised. An example of an error message that is caused by a corrupted ‘eSRL‘ server:

```
java.net.SocketException: Connection reset
        at java.net.SocketInputStream.read(SocketInputStream.java:196)
        at java.net.SocketInputStream.read(SocketInputStream.java:122)
        at java.net.SocketInputStream.read(SocketInputStream.java:210)
        at java.io.DataInputStream.readBoolean(DataInputStream.java:242)
        at ixa.srl.SRLClient.main(SRLClient.java:70)
Completed: module /usr/local/share/pipelines/nlpp/bin/eSRL; result 0
```

The problem can be solved by shutting down the `eSRL` server. To this end there is a script `kill_eSRL_server` that does this, provided the issuer belongs to the “sudo” group.

This script does the following: First it finds out which process listens on the port on which `eSRL` listens, i.e. port 5005. The it checks whether the command-line instruction for that process is indeed command to start the eSRL server. If that is the case, the script kills the process. The Next time that `nlpp` runs it will automatically start a new `eSRL` server.

To find out whether a process listen on port 5005 and, if so, what the ID of that process is, the script issues the `netstat` command. This command produces a line that looks like

```
tcp 0 0 0.0.0.0:5005 0.0.0.0:* LISTEN 29891/java |
```

The process-id of the listener is located before the string `/java`. The following `awk` script extracts that number:

⟨ *define awk-script to extract eSRL process-id* 1 ⟩ ≡

```
awkscript='{match($7, /([[:digit:]]+)\/[.]*/, arr); print arr[1]}'
```

◇

Fragment referenced in 2.

The command that starts the `eSRL` server has form like:

```
java -Xms2500m -cp \
/usr/local/share/pipelines/nlpp/modules/EHU-srl-server/IXA-EHU-srl-3.0.jar ixa.srl.SRLServer en
```

The command can be read from e.g. `/proc/29891/cmdline`. The script looks whether the string `SRLServer` appear in that line.

"bin/kill_eSRL_server" 2≡

```
#!/bin/bash
⟨ define awk-script to extract eSRL process-id 1 ⟩
procnum=‘netstat -tulpn 2>/dev/null | grep 5005 | gawk "$awkscript"‘
grep SRLServer /proc/$procnum/cmdline
res=$?
if
    [ $res == 0 ]
then
    echo process found: $procnum
    sudo kill $procnum
else
    echo eSRL process not found
fi
```

◇

2 Web-resources

2.1 Implement a demo-app

The following is actually a script in `cltl.nl`, not in `kyoto.let.vu.nl`. It serves to install a flask app in `demo.nl`.

It is possible to install a flask app in such a way, that it can be reached as e.g. `demo.cltl.nl/my_app`. There are certain restrictions to this method:

- The app must run on Anaconda as installed in `/usr/local/share/anaconda`. However, it is possible to generate a virtual environment that is derived from this Anaconda application.
- It must be made sure that internal references to the app use the correct URL, including the `my_app` subdirectory.

To install such an app, there exists the following script `add_flask_demo`. The script works as follows:

- It generates a `wsgi` script in directory `/usr/local/demo_wsgi` that activates the app.
- It generates an instruction in the configuration file for site `demo.cltl.nl` that installs the `wsgi` script as `WSGIScriptAlias`.

Suppose you have a Flask app `appdir/my_app.py` and you want to connect it to URL `demo.cltl.nl/the_app`, then invoke the script with the following command:

```
sudo add_flask_demo -n the_app appdir/my_app
```

The script will perform the following:

1. Generate a `wsgi` script in directory `/usr/local/share/demo_wsgi` that starts your script.
2. Include a `WSGIScriptAlias` in the configuration file for the site `demo.cltl.nl` (i.e. `/etc/apache2/sites-available/demo_klipperaak.conf`).

What the script does is described in the help function:

⟨ *help function of add_flask_demo* 3a ⟩ ≡
```
#Help function
function HELP {
  echo -e \\n"${SCRIPT} -- Publish a Flask demo on ${UL}demo.cltl.nl${NORM}"\\n
  echo -e "${BOLD}Usage:${NORM}"
  echo -e "${SCRIPT} [ -n NAME ] [ -v VENV ] FILE"
  echo -e "${SCRIPT} -h"\\n
  echo -e "FILE   Python script with flask app."
  echo -e "-n     Give the demo another name than that of the Python file"
  echo -e "-v     Use virtual environment VENV instead of host environment."
  echo -e "-h     Display this help message."\\n
  echo -e "Example: ${BOLD}$SCRIPT -n super -v mydemo/venv mydemo/mydemo.py ${NORM}"
  echo -e "         publishes app in ${UL}mydemo.py${NORM} as  ${UL}demo.cltl.nl/super${NORM}"\\n
}

◇
```
Fragment referenced in 8a.

2.1.1 Set up the app

Generate a `wsgi` file that performs the following:

- Activate a virtual environment if that is requested by the option `-v`;
- Add the directory of the app to the `PATH` variable;
- Invoke the app.

If the user provided a `-v` option, variable `virtenv` has been set. From Stack-overflow we learn that the best way to check whether a variable `var` has been set is to test expression "`! -z ${var}`".

When a virtual environment ought to be used, have the `wsgi` script to execute the full path to the Python script `activate_this.py`. I am sorry, but I forgot from where I stole this code.

⟨ *build the wsgi file* 3b ⟩ ≡
```
rm -f $WSGI_DIR/$wsgi_filename
if
  [ ! -z ${virtenv} ]
then
  virtenv_full="$( cd $virtenv && pwd)"
  ⟨ wsgi-line (3c activate_this = '$virtenv_full/bin/activate_this.py' ) 3f⟩
  ⟨ wsgi-line (3d with open(activate_this) as file_: ) 3f⟩
  ⟨ wsgi-line (3e     exec(file_.read(), dict(__file__=activate_this)) ) 3f⟩
fi
◇
```
Fragment defined by 3b, 4a.
Fragment referenced in 8a.

Note that writing quotes with the Bash `echo` command is difficult. To write a string that contains single quote characters (`'`), I found out that it works to wrap the string to be written in double quote characters (`"`) and not escape the single quote.

⟨ *wsgi-line* 3f ⟩ ≡
```
  echo "@1" >> $WSGI_DIR/$wsgi_filename
◇
```
Fragment referenced in 3b, 4a.
Uses: `WSGI_DIR` 7a, `wsgi_filename` 7a.

Start the app in the `wsgi` file.

⟨ *build the wsgi file* 4a ⟩ ≡
⟨ *wsgi-line* (4b `import sys`) 3f⟩
⟨ *wsgi-line* (4c `sys.path.insert(0, '$demo_dir') )` 3f⟩
⟨ *wsgi-line* (4d `from $demo_filename_without_py import app as application`) 3f⟩
◇

Fragment defined by 3b, 4a.
Fragment referenced in 8a.
Uses: `demo_filename_without_py` 6c.

Add a `WSGIScriptAlias` statement to the site-config-file for Apache. In this file there is a line with text `Here the WSGIScriptAliases`. Put the `WSGIScriptAlias` right under this line. The following `AWK` script performs this:

Modify the Apache config-file. This has to be done carefully. If something is wrong, Apache might not work anymore causing all the sites that this host supports to drop out of the air. Therefore we proceed as follows:

- Copy the existing Apache config-file to a temporary directory;
- Generate a modified config file and replace the original config-file with it;
- Try to restart Apache. If this fails, restore the original config-file, restart Apache and write a "failure" message.

⟨ *add item in Apache site-config-file* 4e ⟩ ≡

```
tempdir=‘mktemp -d -t flas.XXXXXX‘
cp  $sitesdir/$siteconfigfile $tempdir/$siteconfigfile
gawk '{ print }
      /Here the WSGIScriptAliases/ {
        printf( "    WSGIScriptAlias /%s %s/%s\n", demo_name, WSGI_DIR, wsgi_filename )
      }
      ' \
      demo_name=$demo_name WSGI_DIR=$WSGI_DIR  wsgi_filename=$wsgi_filename \
      <$tempdir/$siteconfigfile  >$tempdir/new.$siteconfigfile
sudo cp $tempdir/new.$siteconfigfile   $sitesdir/$siteconfigfile
◇
```

Fragment referenced in 8a.
Defines: `new.siteconfigfile` Never used, `siteconfigfile` 4f, 7ab, `sitesdir` 4f, 7a, `tempdir` 4f.
Uses: `WSGI_DIR` 7a, `wsgi_filename` 7a.

⟨ *restart Apache* 4f ⟩ ≡

```
service apache2 reload
result=$?
if
   [ $result -gt 0 ]
then
  cp $tempdir/$siteconfigfile $sitesdir/$siteconfigfile
  service apache2 reload
  echo "Error. App not installed. Sorry." >&2
  exit $result
fi
rm -rf $tempdir
◇
```

Fragment referenced in 8a.
Uses: `siteconfigfile` 4e, 7a, `sitesdir` 4e, 7a, `tempdir` 4e.

2.1.2 Set the parameters

As we have seen, a few parameters that have to be set are involved:

Name	default	explanation
`virtenv`	unset	
`WSGI_DIR`	`/usr/local/share/demo_wsgi`	to store `wsgi` files
`wsgi_filename`	Name of python script	WSGI script.
`demo_dir`	unset	Dir. of app.
`demo_filename_without_py`	unset/option	Path to app without suffix
`sitesdir`	`/etc/apache2/sites-available`	Dir. for site config files
`siteconfigfile`	`demo_klipperaak.conf`	Apache configuration file
`demo_name`	demo_filename_without_py	suffix of URL

The user can set `virtenv` with command-line option `-v` and she may specify a name of the demo (i.e. the suffix of the URL, e.g. `demo.cltl.nl/suffix`) that would otherwise be set to the name of the file with the Python script of the app.

The macro below interprets the command-line options using the `getopts` mechanism (see this tutorial in Stackoverflow).

⟨ *get the options of add_flask_demo* 5 ⟩ ≡

```
unset demo_name
unset virtenv
while getopts :n:v:h opt
do
    case $opt in
        n)
            demo_name=$OPTARG
            ;;
        v)
            virtenv="$(cd "$(dirname "$OPTARG")"; pwd)/$(basename "$OPTARG")"
            ;;
        h)
            HELP
            exit 1
            ;;
        \?)
            echo "unknown option: $OPTARG."
            HELP
            exit 1
            ;;
    esac
done
shift $((OPTIND-1))
◇
```

Fragment referenced in 8a.
Uses: `virtenv` 3b.

The user must specify the path to the Python script with the app. So, let us abort execution and print a message when the user did not do this.

⟨ *get location of the flask app or die* 6a ⟩ ≡

```
if
  [ -z ${1+x} ]
then
  HELP
  exit 1
fi
```

◇

Fragment defined by 6abcd.
Fragment referenced in 8a.

If we have survived the above test, construct the full path to the python script from the command-line argument `$1`. Check whether the app really exists and abort execution otherwise.

⟨ *get location of the flask app or die* 6b ⟩ ≡

```
demo_full_filename="$(cd "$(dirname "$1")"; pwd)/$(basename "$1")"
if
  [ ! -e $demo_full_filename ]
then
  echo Error: $demo_full_filename does not exist. >&2
  HELP
  exit 4
fi
```

◇

Fragment defined by 6abcd.
Fragment referenced in 8a.
Defines: `demo_full_filename` 6c.

Derive the name of the script with the app and the path to the directory in which the script resides:

demo_full_filename: Path to the python file of the app.
demo_filename: Name of the python file itself.
demo_filename_without_py:

⟨ *get location of the flask app or die* 6c ⟩ ≡

```
demo_filename=$(basename $demo_full_filename)
demo_filename_without_py=${demo_filename%.py}
demo_dir=$(dirname $demo_full_filename)
```

◇

Fragment defined by 6abcd.
Fragment referenced in 8a.
Defines: `demo_filename` Never used, `demo_filename_without_py` 4d, 6d.
Uses: `demo_full_filename` 6b.

Set the demo-name to the name of the Python script if the user did not specify a demo-name.

⟨ *get location of the flask app or die* 6d ⟩ ≡

```
if
    [ -z ${demo_name+x} ]
then
  demo_name=$demo_filename_without_py
fi
```

◇

Fragment defined by 6abcd.
Fragment referenced in 8a.
Uses: `demo_filename_without_py` 6c.

Set the name and the path of the `wsgi` script that invokes the app and set the path to the Apache config-file:

⟨ *set parameter values for add_flask_demo* 7a ⟩ ≡

```
WSGI_DIR=/usr/local/share/demo_wsgi
wsgi_filename=$demo_name.wsgi
sitesdir=/etc/apache2/sites-available
siteconfigfile=demo_klipperaak.conf
```

◇

Fragment defined by 7ab.
Fragment referenced in 8a.
Defines: `siteconfigfile` 4ef, 7b, `sitesdir` 4ef, `WSGI_DIR` 3bf, 4e, 8a, `wsgi_filename` 3bf, 4e, 8a.

Finally, check whether an app with the chosen name does not yet exist.

⟨ *set parameter values for add_flask_demo* 7b ⟩ ≡

```
grep -q "WSGIScriptAlias[[:space:]*]/$demo_name" /etc/apache2/sites-enabled/$siteconfigfile
if
  [ $? == 0 ]
then
  echo "Error: Demo $demo_name exists already" >&2
  exit 5
fi

```

◇

Fragment defined by 7ab.
Fragment referenced in 8a.
Uses: `siteconfigfile` 4e, 7a.

2.1.3 Putting everything together

Finally, produce the script:

"bin/add_flask_demo" 8a≡

```
#!/bin/bash
# add_flask_demo -- publish a flask demo.
# Argument: Python-file with app
# Options: -n Alternative name for demo
#          -v path to virtual environment.
#Set Script Name variable
SCRIPT=‘basename ${BASH_SOURCE[0]}‘
⟨ pretty fonts for help function 8b ⟩
⟨ help function of add_flask_demo 3a ⟩
⟨ get the options of add_flask_demo 5 ⟩
⟨ get location of the flask app or die 6a, ... ⟩
⟨ set parameter values for add_flask_demo 7a, ... ⟩

#
# Break if a demo with the same name exists already.
#

#
# Generate wsgi file
#
rm -f $WSGI_DIR/$wsgi_filename
⟨ build the wsgi file 3b, ... ⟩
⟨ add item in Apache site-config-file 4e ⟩
⟨ restart Apache 4f ⟩


◇
```

Uses: WSGI_DIR 7a, wsgi_filename 7a.

3 Miscellaneous

3.1 Fonts

The help function uses pretty fonts. I forgot where I stole these font declarations.

⟨ pretty fonts for help function 8b ⟩ ≡

```
#Set fonts for Help.
NORM=‘tput sgr0‘
BOLD=‘tput bold‘
REV=‘tput smso‘
UL=‘tput smul‘
◇
```

Fragment referenced in 8a.

4 Indexes

4.1 Filenames

4.2 Macro's

⟨ add item in Apache site-config-file 4e ⟩ Referenced in 8a.
⟨ build the wsgi file 3b, 4a ⟩ Referenced in 8a.
⟨ define awk-script to extract eSRL process-id 1 ⟩ Referenced in 2.
⟨ get location of the flask app or die 6abcd ⟩ Referenced in 8a.
⟨ get the options of add_flask_demo 5 ⟩ Referenced in 8a.
⟨ help function of add_flask_demo 3a ⟩ Referenced in 8a.
⟨ pretty fonts for help function 8b ⟩ Referenced in 8a.
⟨ restart Apache 4f ⟩ Referenced in 8a.
⟨ set parameter values for add_flask_demo 7ab ⟩ Referenced in 8a.
⟨ wsgi-line 3f ⟩ Referenced in 3b, 4a.

4.3 Variables

`demo_filename`: 6c.
`demo_filename_without_py`: 4d, 6c, 6d.
`demo_full_filename`: 6b, 6c.
`siteconfigfile`: 4e, 4f, 7a, 7b.
`sitesdir`: 4e, 4f, 7a.
`tempdir`: 4e, 4f.
`virtenv`: 3b, 5.
`WSGI_DIR`: 3bf, 4e, 7a, 8a.
`wsgi_filename`: 3bf, 4e, 7a, 8a.

Index