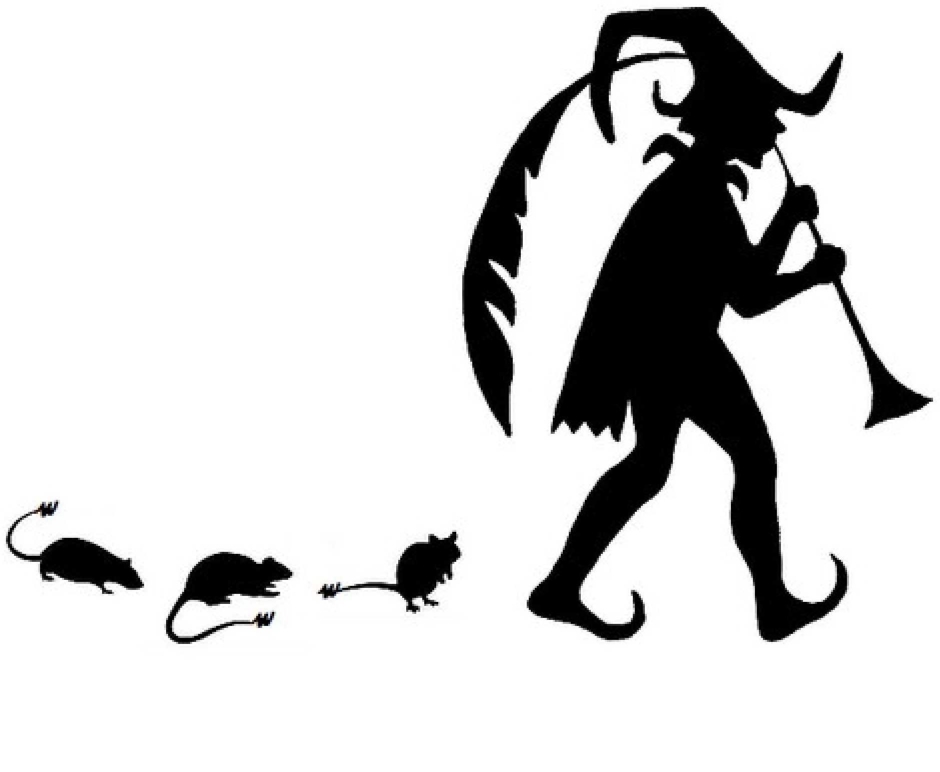
pipEdit
V.0.1

Users guide



the_piper@web.de
2019-08-11

Inhaltsverzeichnis

1. License

2. Introduction

pipEdit is meant to be a lookalike editor of the ISPF editor IBM offers on their mainframes, it is not meant to be a 100% clone of it or of the whole ISPF environment.

The plan is to create (and have) an editor which has a similar feeling like the IBM one, but some things might be different.

For example, I haven‘t found out right now how to make a difference between the ENTER key and the ENTER key on the number pad, so one can be the XMIT/SEND key and the other won‘t.

That, and some other stuff, are some minor differences.

It can be compiled with ncurses and, if not available, with my tiny ncurses replacement, pipcurses, which should work with all vt52, vt100, ansi, xterm terminals.

3. Compiling pipEdit

pipEdit ist developed with gcc, the Gnu C compiler under Linux.

To compile it with ncurses type:

```
gcc pipedit.c -opipeditlc.bin -O -lcurses
```

Without ncurses, using my ncurses replacement, pipcurses, type this:

```
gcc pipedit.c -opipedit.bin -O -D__USE_PIPCURSES__=1
```

Of course you can try to compile it for Windows, Dev-Cpp, for example, is a free compiler which should be able to compile source code for gcc.

https://www.bloodshed.net/devcpp.html

4. Configuring pipEdit

pipEdit uses a configuration file to store/read it‘s configuration data, named pipedit.cfg

4.1 Environment variable

To know where pipedit.cfg is located, the editor reads an environment variable named $PIPEDITCFG which holds the complete path to the config file.

Example:

```
PIPEDITCFG="/home/myname/pipEdit/pipedit.cfg" ; export PIPEDITCFG
```

Define this variable in .bashrc, .kshrc or whereever it suits you.

4.2 Configuration file

The configuration file itself looks like this:

```
####################################
# Global configuration for pipEdit #
####################################
# target is used in *.par files to have multiple configurations
# for multiple compilers
#
target=GNUCOBOL
###target=MICROFOCUS
#
# Where are the macros located the edtior can use?
#
macros=/home/myname/pipEdit/macros
#
# language overwrites the environment variable $LANG
# This can be used for language files for the editor, macros and
# so on.
# If left blank, $LANG is used, if $LANG isn't defined "en" is
# used.
# default means, no file is read, the default values of the editor
# are used.
#
###language=default
language=
#
# various translations of texts and messages
#
langfiles=/home/myname/pipEdit/langfiles
```

5. Backups

pipEdit creates backups of the file being edited. Just to be safe…..

The backups are stored in the `/tmp` folder and are named, for example, we are editing the file hello.cob:

`hello.cob.grandfather`	the oldest backup
`hello.cob.father`	the 2nd oldest backup
`hello.cob.son`	the current backup

As you might guess, every editing session, the grandfather is overwritten with the father, the father with the son and the son with the current file before being edited.

6. Commands

save
save4macro
res / reset
can / cancel
cols
l / loc
f / find

7. Line commands

d
i
r

8. Macros

pipEdit supports macros.

A macro is just a program, script, whatever, which is called by pipEdit, getting a defined number of parameters and reads and modifies a text file.

8.1 Parameters for macros

8.1.1 File name

This is the name of the temporary file pipEdit writes before calling the macro and reads after the macro is done.

The macro changes this file, like the comp macro, which inserts message lines of the error messages of the compile into the source code.

The format of this file is this (and the result pipEdit reads, must be the same format):

Bytes	**Type**	**Content**
0-5	Line number	6 digits line number
6	Line type	I=inserted line, M=message, E=Error, N=normal line, X=eXcluded line
7-nnn	Text	The content of the line

The line number itself is ignored, when pipEdit reads the result of the macro again. The line number is used by the comp macro to find the right line in the source code, where the error messages are displayed.

8.1.2 Parameter file

A parameter file, `<filename>.par`, looks like this (this is a file for the comp macro):

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

```
### M i c r o f o c u s
#####################################################
@target=MICROFOCUS
pre=#!/bin/bash
pre=. /adm/config/basis.prof
pre=export COBCPY=$HOME/Projects/cpy
###-----------------------------------
compiler=cob
options=-x -P
binary=hello.bin
listing=hello.lst
movebinto=/home/yourname/yourlocation/Local_bin/
movelstto=/home/yourname/yourlocation/Listings/
removetmp=.idy .int .cs9Filename
###post=rm *.idy *.int *.cs9
#####################################################
### G n u C o b o l
#####################################################
@target=GNUCOBOL
pre=#!/bin/bash
compiler=cobc
options=-x -Thello.lst
binary=hello.bin
listing=hello.lst
movebinto=/home/yourname/yourlocation/Binaries/
movelstto=/home/yourname/yourlocation/Listings/
###-----------------------------------
post=exit
post=# And thats it
```

Macros can read this file, in this case the comp macro.

This file specifies how to compile COBOL programs for Microfocus or GnuCobol.

Which compiler is used is specified with the `@target=` tag.
`@target=MICROFOCUS` or
`@target=GNUCOBOL`

All lines following the right `@target=` tag will be used by the comp macro to compile the current source code.

The `@target=` itself is specified in the `pipedit.cfg` file.

The `pre=` lines are written at the beginning of the compile shell script.

The `post=` lines are written at the end of that script.

Between those lines the comp macro generates code for the compile from the tags

`compiler=`	name of the COBOL compiler
`options=`	options for the compiler
`binary=`	name of the output, the binary
`listing=`	name of the listing file
`movebinto=`	Where the result of the compile, the binary, is moved to
`movelstto=`	where the listing file should be moved to

This is an example of a *.par file for the comp macro.
When you write your own macros, you will create your own parameter files, fitting to the needs of the macros.

8.1.3 Filename

The original name of the file currently being edited.

8.1.4 Config file

This is `pipedit.cfg`, or however you name it in the environment varialble, the macro can read that config file too and use it‘s values.

Those parameters are given to the macro, the macro can use them to read configuration values from the *.par file or from the config file, process and modify the termporary file, and thats it right now.

That is how macros work with pipEdit.

Further plans: Return something like error messages displayed where pipEdit displays it‘s own error messages, relocating the cursor and such.

9. comp Macro

The comp macro is a macro, which compiles the (COBOL) source coded loaded into pipEdit and displays error messages as message lines right into the source code.