toolselect — transparent access to tool versions

A.T. Hofkamp

Id: userman.tex 11 2007-01-05 15:46:21Z hat

Abstract

In many occasions, it is useful to have several versions of a tool installed at a system. It is useful for development, for example checking behavior of some feature between different versions of a tool. Also, it may be the case that different users want or need to use a different version of the tool for some reason.

At the same time, having several versions of a tool at a system can be a source of trouble. Users have to setup their account such that they use the right version. This often includes the need to setup certain environment variables. Failing to do the setup properly, in particular when switching from one version to another, may be the source of subtle tool problems. In addition, users need to be informed of availability of new versions.

The `toolselect` package aims to simplify the life of the system administrator and users by providing a simple solution to selecting a tool version. An administrator can configure the general pattern to find tool versions, as well as specifying the required setup. Users get a simple frontend to select a version, the `toolselect` package takes care of all the setup details for them.

1 Introduction

There are many possible reasons why more than one version of a piece of software (a *toolset*) may be installed at a single system. At system level, different users may want to use different versions of the same software. At user level, you may want to experiment how different versions of some locally created or locally installed software react to some specific situation. Developers may want to have several experimental versions installed in order to track down a problem.

Most software allows to have different versions installed next to each other, you can install each version with a different prefix directory. The catch is however that at some point in the chain, you as user must make a decision which version you want to use, and act accordingly. You need to ensure that you execute the right version, most often either by providing an absolute path[1] with some indication of the version in it, or by the name of the executable extended in some way with the version you want to run (for example, 'python2.4'). Often, besides starting the correct binary, you also need to set some environment variable(s) to match the selected version otherwise the software uses a mix of data and code from different versions with may result in all kind of subtle or not-so-subtle problems.

Configuring your account to statically use a specific version is not terribly difficult. Set up the environment variables in the startup scripts of your shell, for example in the `.bash_profile` or the `.bashrc` files if you use the *bash* shell. Also, create an alias or make a (soft-)link to the program-version you want to use in you personal `bin` directory[2]. Statically setting a version is a

[1]An absolute path is a path to a file that starts at the root of the file system, for example '/usr/bin/grep'.

[2]Usually, `$HOME/bin` .

bit messy (editing scripts, hard-coding paths in files in your account), once you have configured it, it will work. That is, until you want to switch to another version or the version you use ceases to exist (to make room for newer versions, or due to license expiration). If you want to switch you need to do the entire setup again, this time without leaving pointers to the old version. Unfortunately, the last time you setup your account was ages ago, so you don't remember what you need to do.

The situation gets worse if you (need to) switch often between versions, for example to compare behavior of different versions with each other, or to test a new version. In that situation, manually seting up the environment variables, links, and aliases each time is just not practical. This is often solved by scripting away the setup modifications. In that way, the problem of needing to specify a version shifts from logging-on (with the static solution) to the preparation phase of running the software (which can be changed each time before running).

Developers in the latter situation often have enough knowledge to construct a solution, but many users are not that fortunate. The *toolselect* package aims to solve software selection problems for those users (and for lazy develpers too) in a general way to make it useful for a large class of software. Features of toolselect for a user are

- Easy overview and selection of available versions
- No need to add lines to startup scripts of shells
- Environment variables are setup correctly each time
- You can add your own software packages

Features of toolset for a developer or system administrator are

- Software packages (toolsets) can be installed anywhere at the system
- Toolsets administration can be distributed, to allow different persons managing different parts of the software packages
- New versions can be installed without having to update toolselect files
- Adding a new toolset can be done without updating user files

In Section 2, making the `toolselect` software package ready for use (for a user) is discussed. The next section (Section 3) explains how new toolsets can be added and how to select a version from a toolset. Removal of versions is explained in Section 4. Finally sections 5 and 6 look under the hood of the package, and provide details of the configuration for reference.

Thanks

The toolselect program was much inspired by the *Combinator* tool of *Divmod*[3] which does a similar job as toolselect for branches and trunk in an *svn*[4] environment. The latter allows easily switching between branches and trunk working copies in several svn-based development projects. In addition, it also provides commands to create and merge branches. The latter commands are especially useful for the branch-based development methodology called *UQDS*[5] also from *Divmod* which we have been using for some time now, and seems to work very well for us.

The main differences between both programs are the different application areas (user programs versus svn-based development projects), and audiences (normal users versus developers).

[3]http://divmod.org/

[4]Subversion, a version control system, see http://tigris.org/subversion .

[5]Ultimate Quality Development System.

2 Bootstrapping *toolselect*

After installing the toolselect program, you need to configure the program for your account, and make selections between versions of software packages, so called *toolsets*.

The toolselect package uses a directory in your search path to put its hooks to selected versions of the software under management by toolselect. In particular, since the program needs to make changes in the directory, you need to have write access to the directory. The user bin directory is usually a good candidate, but you can also make a seperate directory for toolselect and add it to the search path in a startup script of the shell. In both cases, toolselect will not overwrite files (or rather, links) it did not create, so it is safe to add your own files to the directory as well (except that it may prevent toolselect from adding selected software if there is a name clash).

To make this manual accessible, concrete examples of commands that are to be entered are shown. For this reason, it is assumed that the toolselect program is installed at /usr. There is a user Sarah with home directory /home/sarah and she is using her own /home/sarah/bin directory. The examples are intended for the Bash shell, but they are so basic that they will work for most shells.

For your system, the paths or shell are likely to be different, but you should be able to adapt the examples to your situation.

The first time Sarah uses toolselect, she needs to state which directory the program should use for putting its links in (the *rerouting directory*). That is done with

```
$ toolselect -r /home/sarah/bin
```

The path entered should be an absolute path (that is, a path starting with a '/'). Note that most paths entered should be absolute paths. In that way, toolselect will work no matter where in the file system you are working. After this command, Sarah has a new /home/sarah/.toolselectrc file with a '[config/toolselect] section, and a line stating the just entered rerouting directory.

While toolselect now works (that is, Sarah can type **toolselect** and the prompt returns without reporting an error), it is not doing very much at the moment, since there are currently no toolsets to choose versions from. (In fact, since toolselect by default lists available toolsets, outputting nothing before the prompt returns confirms that there are indeed no toolsets.) In the next section, this shortcoming will be addressed.

3 Adding toolsets

A toolset is a number of different versions of the same tool. Often, one tool has multiple commands. It also may need environment variables to have a specific value (which changes with each version).

For each toolset, toolselect needs a section in a configuration file. This may be in her own /home/sarah/.toolselectrc file, but it may also be a file elsewhere at the file system. Normally, the person installing versions of the toolset also takes care of creating a file with the necessary information. Here, it is assumed that somebody else has prepared a configuration file section for a toolset already. The next section will explain what to do when this configuration does not exist.

3.1 Adding an existing toolset

To make the program easy to try (and to make the manual easier to follow), the toolselect distribution comes with the infamous 'hello world' program in two glorious versions. At the system Sarah is using, it is located at /usr/share/toolselect/examples/helloworld. To use this toolset, she only has to give the location of the configuration file (which, by convention, has a '.toolselect' extension). She enters

```
$ toolselect -c "/usr/share/toolselect/examples/helloworld/helloworld.toolselect"
```

While the double quotes around the path are not needed in this case, the argument is assumed to be a pattern matching toolselect configuration files rather than a singl file. In the general case, Sarah would need to use quotes to protect the pattern.[6]

Toolselect stores the provided pattern, and outputs a list of toolsets that it has found:

```
toolset 'helloworld' (at /usr/share/toolselect/example/helloworld):
    version hello-1 (not selected)
    version hello-2 (not selected)
```

Not entirely surprisingly, Sarah now has toolset 'helloworld'. It has two versions, namely 'hello-1' and 'hello-2'. Both of them are currently not selected. Sarah can selects the first version by entering:

```
$ toolselect select helloworld hello-1
```

To verify the selection, Sarah runs toolselect again:

```
$ toolselect list
```

(since listing the toolsets is the default action, you can omit the 'list' command above). Toolselect responds with

```
toolset 'helloworld' (at /usr/share/toolselect/example/helloworld):
    version hello-1 selected
        command 'hello' (pointing to /usr/share/toolselect/example/helloworld/hello-1/hello)
    version hello-2 (not selected)
```

Version 'hello-1' has been selected, and the program also lists the commands it has set up and what each command actually executes. To test it, Sarah enters **hello**, and the system responds with

```
Hello world (version 1)
HELLO_DATA = /usr/share/toolselect/example/helloworld/hello-1/datafile
```

which is the expected output, the version is correct (notice the '1' at the end of the first line) and the environment variable 'HELLO_DATA' is set to the data file in the hello-1 directory.

Sarah can now use the version 1 'hello' program. Switching to the other version is a matter of running another 'toolselect select' command.

3.2 Doing it yourself

While Sarah was lucky to have a '.toolselect' configuration file available, your luck may just have run out, and you need to write your own file. Luckily, it is not as difficult as it may seem. To demonstrate this, below is the contents of the helloworld.toolselect configuration file:

[6] In other words, **"/usr/share/toolselect/examples/helloworld/*.toolselect"** will also work. This is useful if you have a directory with a collection of toolselect configuration files. You can also use **"/usr/share/toolselect/examples/*/*.toolselect"** if you like which finds all toolselect configuration files hidden in a subdirectory of some (in this case examples) directory.

```
[toolset/helloworld]
installedbasepath = /usr/share/toolselect/example/helloworld
versions = hello-*
executables = hello
environment =
    HELLO_DATA = $(base)/$(version)/datafile
```

The name of the toolset is in the section header (‘[toolset/helloworld]’), prefixed by ‘toolset/’ to indicate the type of the section. On the lines below the section header are the options of the section. The first option is the absolute base path to the toolset, the common prefix path to all versions of the tool. Each version is assumed to be in a seperate subdirectory. The ‘versions’ option is a list of patterns relative to the common prefix, in this case one simple pattern ‘hello-*’. The executables option is a list of patterns relative to a single version that match commands to execute. In this case, there is just one ‘hello’ executable immediately in the root of each version. In most cases, a pattern like ‘bin/*’ is more appropiate. Finally, the ‘environment’ option need only exist if the tool requires setting (prefixing) of values in environment variables. Each environment variable should be put on its own (indented) line. The name of the environment variable is put first, followed by an equal sign, and finally the (prefix) value of the variable. The value is assumed to be a path, and copied literally except for the ‘$(base)’ string which is replaced by the value of the ‘installedbasepath’ option, and the $(version)’ string which is replaced by the version actually chosen (that is, an instance of the ‘versions’ pattern existing at the system). When a command of the tool is executed, the values of the environment variables are inserted as the first path.

As you can see, the definition of a toolset is only a few lines. You can either create a seperate file for each toolset, or have several toolset definitions together in one file. The only requirement is that toolselect reads the data. Adding the section(s) to your own .toolselectrc file is the easiest way to ensure that, otherwise you must provide an additional configuration file pattern (starting with a ‘/’) to toolselect[7] that matches the file(s) containing the toolset section(s).

4 Removing a version

While it may not happen frequently, removal of a version of a toolset is also possible. For example, when Sarah does not want to have the ‘hello’ command any more, she can enter

```
$ toolselect remove helloworld
```

Removal of a toolset is currently not possible from the command line. After removing the version (shown above), you can remove a toolset by editing the .toolselectrc file, and deleting the path to the configuration file from the ‘extraconfigs’ option.

5 A peek under the hood

While the function of the toolselect software package may seem quite complicated at first, its operation is not very complicated. In this section we take a peek under the hood and show how it works. Also, the layout of the .toolselectrc file is explained both for fun and as a recovery tool that may provide a way out of trouble on a very dark day. The complete and compact reference of the configuration can be found in Section 6.

[7]Either by using the **-c** option again (don’t forget to quote the pattern!), or by editing the .toolselectrc file and adding the absolute path pattern to the ‘extraconfigs’ option.

The first thing that needs to be explained is that the toolselect software package is not one program, it contains two programs. The first (hidden until now) program is the 'pretender' program. After a user has selected a version, the pretender sets up the environment variables, and starts the actual tool that needs to be run (within the selected version). To do its work, it needs a configuration file (the .toolselectrc configuration file) that states what to do. The 'toolselect' program manages the links to the pretender program in the rerouting dir, and allows for easy modification of the .toolselectrc configuration file. In other words, you do not need 'toolselect' to use the tool selection software. Without it, you just need to take care of more details (and these are explained here, so after reading this section you could remove the 'toolselect' program).

To illustrate the internal operation, below is the configuration file (/home/sarah/.toolselectrc) of Sarah after selecting version `hello-1` of the helloworld toolset (slightly shuffled and broken into pieces to make the explanation of its contents easier).

```
[config/toolselect]
reroutingdir = /home/sarah/bin
extraconfigs = /usr/share/toolselect/example/helloworld/helloworld.toolselect
```

The configuration of the toolselect program itself is in the '[config/toolselect]' section. It has two options, namely 'reroutingdir', the absolute path to the rerouting directory (that is, the directory where the 'hello' command is linked). The 'extraconfigs' option is a colon-seperated list of absolute file patterns that match additional toolset configuration files.

```
[toolset/helloworld]
currentversion = hello-1
```

Toolselect also keeps track of what Sarah has selected. In addition to the base path, versions, environment, and executables patterns, toolsets with a selected version have an additional option 'currentversion', which lists the currently selected version of the toolset. There is also an 'installdate' option that records the time of selecting the version, but that option is not shown here.

(Note that the other options of the '[toolset/helloworld]' section are still retrieved from the original /usr/share/toolselect/example/helloworld/helloworld.toolselect file, it is not copied to the .toolselectrc file to allow changes made to the helloworld.toolselect configuration file by the maintainer of the 'helloworld' toolset to appear at the screen of Sarah.)

The link to the 'hello' program of the selected version of the 'helloworld' toolset runs through the 'hello' link in the rerouting directory. Here the pretender program comes into play. Looking at the /home/sarah/bin/hello link:

```
$ ls -l /home/sarah/bin/hello
lrwxr-xr-x 1 sarah sarah /home/sarah/bin/hello -> /usr/libexec/pretender
```

This is the hook to the pretender program. When Sarah types 'hello', she runs the pretender program rather than the 'hello' program from the 'helloworld' toolset. The pretender program also reads the .toolselectrc configuration file, and uses the following information:

```
[command/hello]
executable = /usr/share/toolselect/example/helloworld/hello-1/hello
toolset = helloworld
reroutingpath = /home/sarah/bin/hello
pretenderprogram = /usr/libexec/pretender
```

For each link, a [command/*command-name*] section is created. The 'executable' option is the absolute path to the command of the selected version (the pretender program ultimately runs that file). The 'toolset' option states from which toolset the command comes from. Finally, the 'reroutingpath' and the 'pretenderprogram' options store the entire contents of the /home/sarah/bin/hello link to detect whether the link was created by toolselect or not (toolselect will not delete the link if the contents is not the same).

The environment variables that need to be set are defined in the [environment/*toolset-name*] section. If no environment variables need to be set, the section does not exist. Below if the environment variable section for the 'hello' program:

```
[environment/helloworld]
hello_data = /usr/share/toolselect/example/helloworld/hello-1/datafile
```

Each option has the same name as the name of the environment variable that needs to be set, and the value that it needs to add is at the right (after expanding the '$(base)' and '$(version)' strings).

6 Configuration reference

Below, the used options with their allowed syntax and meaning are listed compactly to use as a reference.

6.1 Pretender configuration

The configuration information used by pretender starts with the name of the command used to start the pretender (through the rerouting). For example, assume the pretender program was started with a command `cmd`. The pretender program looks for a section called 'command/cmd'. In that section, two options[8] are used:

executable The absolute path to the real executable that the pretender program must start.

toolset The name of the toolset that the command ('cmd' in the example) belongs to.

The 'executable' option points to the program that must actually be started when the user enters 'cmd'. In general, this is an absolute path with some version indication inserted somewhere in the path. The 'toolset' option is used to find the environment variables settings required for the command. If for example, the command is part of the toolset 'utils', the pretender program searches for the 'environment/utils' section. All options in that section are assumed to be settings for the command (actually, all commands of that toolset). Environment variables are assumed to be a list of paths, seperated by the path seperator (':' under POSIX). If an environment variable listed in the section already exists, its value is prepended to the existing value, else the environment variable is created.

6.2 Toolselect configuration

Like the pretender program, the tool select uses the toolselect configuration file as a starting point. It uses different sections of the file to do its job however. The configuration of toolselect itself is in the section 'config/toolselect'. Options of this section are

[8]Other options exist in the section, but they are used by the toolselect program rather than the pretender program.

extraconfigs A list of paths (seperated by the path seperator) for adding additional configuration files. Each path may be a pattern (that is, it may contain wildcards), and should point to files which are used as additional configuration files.

reroutingdir The directory to put rerouting links to the pretender program in. For toolselect to work, this directory should be in the search path. It is allowed to add other programs there as well, *toolselect* goes to great lengths to ensure it doesn't delete anything it doesn't own.[9].

The additional configuration files are used oly to find additional software installed at the system. By moving these outside the user configuration file, they can be installed and maintained at a central point at the file system. Toolselect only reads the additional configuration files, write access is not needed.

All configuration files are sections of the form 'toolset/*name*' with *name* the name of the toolset. Together, these sections represent the set of installed tools at the system. In each of these sections, the following options should exist:[10]

installedbasepath An absolute common prefix path for all installed versions of this toolset.

versions A list of path patterns (seperated by the path seperator) relative to the 'installedbasepath' to find installed versions. In general, the combination of 'installedbasepath' and one instance of a version pattern seperated by a directory seperator ('/' at a POSIX system) should refer to the base directory of an installed version of a tool.

environment A list of lines, where at each line a key/value pair describes the value of an environment variable to set. The key part is the name of the environment variable to set, the value part is the value that should be prepended to the variable by the pretender program. In a value, the pattern '$(base)' refers to the common prefix path defined by the 'installedbasepath' option, and '$(version)' refers to an instance of a path pattern defined by the 'versions' option. If no 'environment' option exists, it is assumed that no environment variables need to be set.

executables A list of path patterns relative to the base directory of an installed version (seperated by the path seperator). This defines the set of executable commands that should be installed through the pretender program. If the option is not available, its value is assumed to be 'bin/*' (at a POSIX system).

For its internal administration, the toolselect program also adds an option of its own to the toolset sections, namely

currentversion The currently installed version by toolselect, that is, the currently selected version that can be used through rerouting by the pretender program.

installdate Timestamp (as floating point value in seconds since epoch) of the installation of this toolset by toolselect.

These options are mainly to inform the user. Not having these options set is allowed at the expense of slightly less information for the user.

In addition, the toolselect program also adds two options to the 'command/*cmd*' sections of the pretender program to allow for safe removal of the rerouted commands:

[9]Well, unless there is a bug in that part of the program. . . .

[10]Due to the fact that additional configuration files are considered to be read-only, some options of toolsets defined in additional configuration files may be in the the user configuration file instead.

reroutinglink Absolute path to the link created in the search path (typically the reroutingdirectory of the config/toolselect section).

pretenderprogram Absolute path to the pretender program used, that is, the value of the rerouting link.

These values are used to check that the entries in the rerouting directory are owned by the toolselect program. If they do not correspond with the values at the file system, toolselect will[11] not remove the entry.

[11]Or rather, should.