NAME

apparix – bookmark directories and apparate inside them

SYNOPSIS

Apparix allows you to bookmark directories and later jump to them using the mark. It is possible to directly jump to subdirectories of the target directory. The contributed bash completion code facilitates completion both on bookmarks and subdirectories.

This manual page suffers from an excess in verbosity due to the many examples, explanations of the bells and whistles, and comparisons with other approaches to bookmarking. The fundamental idea is simply that typing a string of your own choosing takes you to the directory associated with it. Apparix does little more than maintaining a list of keys and values. It obtains directory names and listings, associates path names (values) with bookmarks (keys), and has some facilities for manipulating keys and values. The functions involving apparix (`bm`, `to`, and `portal`) provide the user interface.

GETTING STARTED

Install apparix. This should be as easy as *./configure --prefix=$HOME/local && make && make install*, or perhaps a pre-packaged apparix is available for your system. Then get hold of the **to**, **bm** and **portal** shell handles. These are either aliases or functions depending on your shell. Currently csh-style shells and bash are supported. Get the ones you need either from the **FILES** section or by issuing `apparix --shell-examples`. Activate them by simply pasting them in a shell or adding them to the appropriate resource file, e.g. `$HOME/.cshrc` or `$HOME/.profile` (do not forget to *source* the resource file). The handles **to**, **bm** and **portal** can of course be changed to any name desired. The following is a mock-up shell navigation session.

```
   > pwd
   /home/eez/cvs/xyz/tfa/faq/zut/bar/foo
   > ls
   src/ doc/ CVS/ bin/
   > bm xkr        # bookmark as xkr (funny name though)
   > bm            # bookmark as foo (trailing component is default)
(later)
   > to xkr        # cd to /home/eez/cvs/xyz/tfa/faq/zut/bar/foo
(alternatively)
   > to xkr src    # cd to /home/eez/cvs/xyz/tfa/faq/zut/bar/foo/src
(alternatively)
   > to foo        # cd to /home/eez/cvs/xyz/tfa/faq/zut/bar/foo

(later)
   > ls
   aap pyu/ qua tim/ zut/
   > pwd
   /home/eez/another/branch/deep/down/under
   > portal        # bookmark as portal, imports tim zut pyu bookmarks
   added flock of 3 in portal /home/eez/another/branch/deep/down/under

(later)
   > to zut        # cd to /home/eez/another/branch/deep/down/under/zut

(later)
   > apparix    # show all bookmarks
   --- portals
   e               /home/eez/another/branch/deep/down/under
   --- expansions
   j pyu           /home/eez/another/branch/deep/down/under/pyu
   j tim           /home/eez/another/branch/deep/down/under/tim
   j zut           /home/eez/another/branch/deep/down/under/zut
   --- bookmarks
   j xkr           /home/eez/cvs/xyz/tfa/faq/zut/bar/foo
   j foo           /home/eez/cvs/xyz/tfa/faq/zut/bar/foo
```

In the last example apparix simply shows all its bookmarks. The first batch shows portals. The second batch shows secondary bookmarks expanded from portals. The third batch shows all regular bookmarks.

Apparix also allows subdirectory specification. If this is combined with the bash completion code it yields a powerful way of navigating container directories, i.e. directories that contain a large number of subdirectories. Refer to the **subdirectory specification** section.

Further options

[--add-mark (*add jump bookmark*)**]** **[--add-portal** (*add portal bookmark*)**]** **[-sm** <mark> (*squash repeated marks*)**]** **[-sd** <mark> (*squash repeated destinations*)**]** **[-lm** <mark> (*list bookmarks with this mark*)**]** **[-ld** <mark> (*list destinations with mark indirection*)**]** **[-favour** <list> (*duplicate resolution*)**]** **[-purge** pat (*delete bookmarks*)**]** **[-purge-mark** (*pat*)**]** **[-d** (*dump resource file to STDOUT*)**]** **[-l** (*list available jumps*)**]** **[-u** <num> (*remove last <num> additions*)**]** **[--rehash** (*re-expand portal bookmarks*)**]** **[--bu** (*create backup of resource file*)**]** **[-bu** <fname> (*create backup in <fname>*)**]** **[--cwd** (*use getcwd(3), not pwd(1)*)**]** **[--shell-examples** (*output example macros*)**]**

DESCRIPTION

Apparix combines the properties of the *cdargs* utility and the CDPATH shell mechanism for fast navigation through the file system. It is especially useful for visiting and documenting both often- and rarely-used locations. Apparix enables you to attach marks to locations and jump to those locations by loading the mark. Marking, unmarking and jumping are simple operations that are performed in the shell. All actions take effect immediately in all shells running. By setting up convenient aliases for marking and jumping the file system can be navigated in a fast and intuitive manner. The **FILES** section lists aliases for csh-type shells and functions for bash, including the setup to equip the **to** function with argument completion in bash.

This section contains some examples of the most common uses of apparix. **OPTIONS** contains a list of additional options available for listing, pruning, and squashing bookmarks.

NOTES features a brief discussion of the advantages of apparix over other approaches such as setting up aliases for often visited directories, using symlinks, CDPATH, or a combination of these. **HISTORY** explains the difference between cdargs and apparix. The sections **duplicate resolution**, **subdirectory specification**, **tab completion**, **copying and moving files**, and **listing bookmarks** further below are also recommended reading.

Apparix works in a manner similar to cdargs. One usually invokes apparix by using pre-defined aliases. Here they will be called **bm** for bookmark, **portal** for a CDPATH-style bookmark and **to** for initiating an apparition (aka jump). These aliases are found below in the **FILES** section and can also be obtained by issuing

```
apparix --shell-examples
```

Suppose your user name is *eez* and your home directory is `/home/eez`. You often visit a directory called `/home/eez/cvs/xyz/tfa/faq/zut/bar/foo`. This is how to create and use a bookmark for foo

```
/home/eez/cvs/xyz/tfa/faq/zut/bar/foo> bm foo
added: foo -> /home/eez/cvs/xyz/tfa/faq/zut/bar/foo
/home/eez/cvs/xyz/tfa/faq/zut/bar/foo> cd
/home/eez> to foo
/home/eez/cvs/xyz/tfa/faq/zut/bar/foo>
```

If one bookmarks a directory by its trailing component as happened in this case, it is not necessary to specify the mark. By default apparix will use the trailing component as the mark. So

```
/home/eez/cvs/xyz/tfa/faq/zut/bar/foo> bm
added: foo -> /home/eez/cvs/xyz/tfa/faq/zut/bar/foo
```

gives the same result.

Another scenario is where you have some directory that contains a largish number of subdirectories, all of which you would like to have bookmarked. If the subdirectories have distinctive names this can be achieved in one fell swoop by marking the parent directory as a *portal*. This is similar to adding the parent directory to the CDPATH environment variable, except that apparix bookmarks are not part of the cd namespace. It is argued in **NOTES** that this is a good thing. Consider this:

```
/home/cvs/bagger/boemel/mcl/mcl/src> ls
alien/      CVS/        impala/     Makefile.am  README      shmcx/
attic/      giraffe/    lib/        Makefile.in  shcl/       shmx/
contrib/    gmon.out    Makefile    mcl/         shmcl/      taurus/
```

Some of the subdirectories have not-so-distinct names such as *contrib* and *attic*, but they happen to be the directories least visited. Issuing:

```
/home/cvs/bagger/boemel/mcl/mcl/src> portal
[apparix] expanded 1 portal to 12 destinations
```

yields all of the subdirectories as destinations bookmarked by the last component of their path name. Incidentally, directory names such as `CVS` can be explicitly excluded from expansion by setting the environment variable `APPARIXEXCLUDE` appropriately – refer to section **ENVIRONMENT**.

Bookmarks resulting from portal expansion are kept in a separate resource file (see **FILES**). Portal expansions can be recreated by issuing

```
apparix --rehash
```

This is useful to reflect a change in the directory naming structure underneath a portal.

duplicate resolution

Apparix allows identical bookmarks to point to different locations. When asked to visit such a bookmark it will by default present a list of options.

The **-favour** *<list>* option can be used to automate resolution. **<list>** is a sequence of single characters. The order in which they are given denote the order in which resolution rules are applied. This option is typically used in the definition of the **to** function/alias or in the bash completion code.

Duplicates are allowed because it can be useful to overwrite a bookmark with a new location. The old bookmark is kept as a matter of policy. Use **-sm** to explicitly squash duplicates.

l *level*; prefer paths with fewer components.

L reverse of the above.

o *bookmark order*; prefer older entries. Entries appearing earlier in the file are considered older, but the actual date of creating the bookmark is not stored. Refer to **editing bookmarks** for more information.

O reverse of the above.

r *regular first*; prefer regular bookmarks over portal expansion.

R reverse of the above.

If there are still ties after the specified rules have been applied apparix will simply take the first matching option. This behaviour cannot be further specified as the program uses a non-stable ordering routine.

It is an absolute prerequisite that **-favour** is used in the bash completion code. Otherwise completion will fail (for a duplicated bookmark) while apparix is waiting for input. Refer to the tab completion description below.

subdirectory specification

When jumping (apparating) you can specify an additional subdirectory after the bookmark. Apparix will append the subdirectory to the destination.

This is useful for projects with directory nodes corresponding with versions. Assume you have a directory structure such as this:

```
/x/y/z/OpusMagnum/v1/
/x/y/z/OpusMagnum/v2/
/x/y/z/OpusMagnum/v3/
```

It is probably easiest to simply bookmark the OpusMagnum directory in some way (say with bookmark `om`). You can then issue '`to om v2`' to jump to `OpusMagnum/v2`. This is more flexible and maintainable than creating bookmarks `om1`, `om2`, `om3`. One could add OpusMagnum as a portal, but with generic names such as `v1` this is not a very extendible approach.

See also the tab completion description below - it is possible to tab-complete on subdirectories of the apparix jump directory.

tab completion

The bash tab completion code does two things. First, it is possible to tab-complete on apparix bookmarks themselves, showing a listing of all available bookmarks (or iterating through them in cyclic mode, depending on your bash settings). Second, once a bookmark has been given tab completion will list or iterate over all the subdirectories of the directory associated with that bookmark. Specifying a string after the bookmark will limit tab-completion to directories matching the shell-pattern in string. *Very* useful.

Be careful to not remove the **-favour** *list* option from the bash completion code. It is necessary to resolve duplicate bookmarks.

editing bookmarks

Apparix appends new bookmarks to the end of the .apparixrc file. Nothing stops you from editing the file, and this is in fact recommended if for example you need to get rid of a bookmark and neither of **-purge**, **-purge-mark**, **-sd**, **-sm** fulfills your needs. It was an easy design choice not to equip apparix with editor capabilities.

copying and moving files

It is straightforward to copy or move files to locations known by apparix. Examples:

```
BASH and variants
   cp FOO $(apparix zoem)
   mv BAR $(apparix zoem doc)
   mv BAR $(apparix zoem doc)/test

CSH and variants
   cp FOO `apparix zoem`
   mv BAR `apparix zoem doc`/test
```

listing bookmarks

Simply issuing apparix gives you a list of bookmarks grouped into three categories, portals, expansions, and bookmarks. Use the **-d** option to dump the resource file to STDOUT exactly as it is. This can be useful when you intend to use the **-u** *num* option to remove bookmarks or portals that were most recently added.

Use **-l** to list all available jumps without their destinations. The jumps are grouped into expansions resulting from portals and regular bookmarks.

OPTIONS

For bookmarking and jumping apparix is best invoked by using the aliases (tcsh-variants) or functions (sh/bash) listed in **FILES**. Apparix has a few options that are useful for pruning, squashing and rehasing bookmarks. These are best issued by invoking apparix directly.

If you are interested in marks or destinations matching a certain pattern, simply issue apparix without arguments and pipe it through your program of choice.

Unary options (those without arguments) usually start with two hyphens except for standardized options such as **-h**. Options that take an argument can be converted to a unary key=value notation, e.g. **-purge-mark foo** is equivalent to **--purge-mark=foo**.

When invoked without arguments apparix will simply dump its bookmarks.

--add-mark (*add jump bookmark*)
This options expects trailing *[mark [destination]]* argument(s). Both arguments are optional. If a single argument is given it is interpreted as a bookmark name to be mapped to the current directory. If two arguments are given the last argument is taken as the target directory. If no argument is given apparix will enlist the current directory as a target bookmarked by the trailing component of the directory path.

--add-portal (*add portal bookmark*)
This option enlists a directory as a portal and adds all subdirectories as bookmarks. The name of the bookmark is simply the name of the subdirectory. By default the current directory is added as a portal. An optional trailing argument will override this behaviour and instead be interpreted as the portal location.

-sm <mar> (*squash repeated marks*)
Apparix will squash bookmarks with mark `<mark>`. This is useful when a mark points to a versioned project, and the project is updated to a new version and a new directory.

Apparix will by default keep the last one occurring in the resource file (corresponding with **-favour O**). This option respects the **-favour** option if given. Duplicating an already existing mark can be useful when it identifies a project for which the underlying directory changes every once in a while (e.g. the project is downloaded from external sources and comes with version information). It is not strictly necessary to squash bookmarks since **to** functions/macros that are equipped with the **-favour** option will generally resolve duplicate matches.

-sd <mark> (*squash repeated destinations*)
All other bookmarks with the same destination as `<mark>` are removed. This is useful when a given destination has acquired multiple bookmarks and you decide to settle on a favourite.

-lm <mark> (*list bookmarks with this mark*)
It lists all bookmarks `<mark>` (noting that it may point to multiple locations).

-ld <mark> (*list repeated destinations*)
This lists all bookmarks `<mark>` (noting that it may point to multiple locations) and additionally lists all other bookmarks that share the destination with any of the first bookmarks. This allows one to predict the effect of issuing `apparix -sd <mark>`.

-purge pat (*delete bookmarks*)
    This deletes bookmarks where destination matches *pat*. All deleted bookmarks are printed to STDOUT. Thus if you regret deleting a bookmark it is easy to add it back. Portal specifications are never affected.

-purge-mark (*pat*)
    This deletes bookmarks where mark matches *pat*. Portal specifications are never affected.

-d (*dump resource file to STDOUT*)
    Dump resource file to STDOUT.

-l (*list available jumps*)
    List available jumps paragraph-style. Portal specifications themselves are excluded, and regular jumps and jumps resulting from portal expansions are listed under different headers.

-u <num> (*remove last <num> additions*)
    Remove last <num> additions. Portal specifications and regular jumps are treated alike.

--rehash (*re-expand portal bookmarks*)
    Apparix will reread the resource file and reexpand portal locations. Useful if directories have been added, renamed, or removed. Refer to section **ENVIRONMENT** for the effect that the environment variable `APPARIXEXCLUDE` has on portal expansion.

-favour <list> (*set duplicat resolution policy*)
    This option has its own section. Refer to **duplicate resolution**.

--cwd (*use getcwd(3), not pwd(1)*)
    By default aparix uses the program *pwd*(1) rather than the system call *getcwd*(3). On some systems it was found that the latter results in paths that contain machine-specific mount components. Appparix will use *getcwd*(3) when **--cwd** is used.

--shell-examples (*output example macros*)
    This outputs example macros. They are also listed in the **FILES** section though.

--bu (*create backup of the resource file*)
    This creates the backup file in .apparixrc.bu.

-bu fname (*create backup of the resource file*)
    This creates the backup file in *fname*. Use **-d** or **-bu -** to dump to STDOUT.

-h (*show synopsis*)
--apropos (*show synopsis*)
    print synopsis of all options

ENVIRONMENT

APPARIXEXCLUDE
    This variable specifies exclusion behaviour when portals are expanded with the `--rehash` option. It has the following syntax:

```
<[:,][<string>]>+
```

    That is, a list of names with each name preceded by a colon or a comma. A colon indicates that `<string>` triggers exclusion of directory names for which the trailing component is identical to `<string>`. A comma indicates that `<string>` triggers exclusion of directory names for which the trailing component contains `<string>` as a substring. Consider:

```
export APPARIXEXCLUDE=:CVS:lib,tmp         # A - example
export APPARIXEXCLUDE=,                    # B - curiosity
```

The first excludes directory names `CVS` and `lib` and any directory name having `tmp` as a substring. The second example will effectively disable portals, as it speficies the empty string which is a substring of all strings.

APPARIXTAG
This variable, if set, is incorporated into the names of the apparix resource files. By default these are `.apparixrc` and `.apparixexpand`. When APPARIXTAG is set to `<tag>` they become `.<tag>apparixrc` and `.<tag>apparixexpand`. This can be used e.g. to maintain different sets of bookmarks on different host machines.

APPARIXLOG
This variable, if set, is interpreted as the name of a log file. The log file keeps track of all newly added bookmarks and portals without ever deleting anything, in the same format as the `.apparixrc` file. If this variable is not set nothing is tracked.

APPARIXPURGE
This changes the way apparix dumps purged bookmarks to STDOUT. By default they are dumped as command lines that will reimport the bookmarks if issued (i.e. cut and pasted). By setting this variable to 1 purged bookmarks are dumped in the format used in the `.apparixrc` file.

FILES

You should use aliases or functions to make apparix really useful. Get them from apparix by giving it the --shell-examples option, or from further below. Note the fragment that provides **to** argument completion in bash.

$HOME/.apparixrc
This is the primary resource file. There is usually no need to edit it by hand. Sometimes it can be useful to edit by hand to remove an unwanted bookmark; refer to **editing bookmarks**.

$HOME/.apparixrc.bu
Apparix creates a back-up file whenever it is asked to remove entries from it. Refer to **editing bookmarks** for options inducing removal. You can explicitly require a backup to be made by either of **--bu** or **-bu** *fname*.

$HOME/.apparixexpand
This contains bookmarks that are expanded from portals. A portal is simply some directory. The names of all subdirectories are taken as bookmarks that point to those subdirectories. This file can be recreated by issuing

```
apparix --rehash
```

$HOME/.bashrc
$HOME/.tcshrc
$HOME/.cshrc
Add the code you need to the appropriate rc file. The macros and functions below point

cd(1) in the right direction.

```
BASH-style functions
---
function to () {
   if test "$2"; then
     cd "$(apparix "$1" "$2" || echo .)";
   else
     cd "$(apparix "$1" || echo .)";
   fi
   pwd
}
function bm () {
   if test "$2"; then
      apparix --add-mark "$1" "$2";
   elif test "$1"; then
      apparix --add-mark "$1";
   else
      apparix --add-mark;
   fi
}
function portal () {
   if test "$1"; then
      apparix --add-portal "$1";
   else
      apparix --add-portal;
   fi
}
# function to generate list of completions from .apparixrc
function _apparix_aliases ()
{   cur=$2
    dir=$3
    COMPREPLY=()
    if [ "$1" == "$3" ]
    then
        COMPREPLY=( $( cat $HOME/.apparix{rc,expand} | \
                       grep "j,.*$cur.*," | cut -f2 -d, ) )
    else
        dir=`apparix -favour rOl $dir 2>/dev/null` || return 0
        eval_compreply="COMPREPLY=( $(
            cd "$dir"
            \ls -d *$cur* | while read r
            do
                [[ -d "$r" ]] &&
                [[ $r == *$cur* ]] &&
                    echo \"${r// /\\ }\"
            done
            ) )"
        eval $eval_compreply
    fi
    return 0
}
# command to register the above to expand when the 'to' command's args are
# being expanded
complete -F _apparix_aliases to
---
```

```
CSH-style aliases
---
alias to    'cd `(apparix -favour rOl \!* || echo -n .)` && pwd'
alias bm   'apparix --add-mark \!*'
alias portal 'apparix --add-portal \!*'
---
```

More elaborate setups are possible. This CSH-style alias:

```
alias to '(test "x" !=  "x\!*") && cd `(apparix -favour rOl \!* || echo -n .)` || appa
```

lists all available jumps if invoked without arguments.

NOTES

Below follow some comments on other approaches to file system navigation. **HISTORY** explains the difference between the venerable **cdargs** program and **apparix**.

CDPATH is only useful in cases where a given directory has subdirectories with distinctive names. It does not usually scale well when there are more than a few paths in CDPATH.

Some people use aliases to jump to often visited directories. I was one of them for a period of ten years. The fact is, those aliases are cumbersome to create and remove and they clutter up the alias namespace. They can clash with executable names when the alias includes the *cd* part. This sometimes prohibits one from assigning the logical bookmark to a given location, especially when one has a lot of source code locations. They can clash with directory names when the aliases just expand to the location. This again means that sometimes a location cannot be assigned its logical bookmark. I have found that setting *cd* jumps aside in their own namespace improves file system navigation by a large factor.

It is also possible to create symlinks to often visited files. Again, creation and removal of these are cumbersome. One could of course create shell functions with a similar interface to apparix or cdargs to handle the symlink lifecycle. On Linux Weekly News *nix* suggested to put these symlinks in a single directory and add that directory to CDPATH. This is quite a neat trick and effectively creates a bookmark navigation system.

Still there are problems with the above approach. One problem with the symlink approach is that they are a bit awkward to edit. One could make a utility to wrap around the problem, but in the end the directory-with-symlinks would functionally be the same as apparix's **.apparixrc** resource file, only more of a kludge. Another problem is that symlinks are awkard when traversing the file system. They confuse the notion of parent directory and '`cd ..`' mostly does the unexpected. Sometimes '`..`' has a different meaning to **cd** than it has to another application, as one will trace back symlinks and the other will not. Finally, a minor objection is that I find it convenient to have bookmarks in a separate namespace than that of *cd*(1). Jumps are magical and it is natural to invoke them by a different method. This is in fact how apparix acquired its CDPATH behaviour. I used CDPATH to jump to a few particular source directories with distinct names that lay deeply hidden in some CVS directory. Once I started using apparix however, I would mistakenly issue *to* rather than *cd* to jump to those locations. My brain classified both types of jump in the same category.

Apparix (and cdargs) have another use besides jumping, namely annotation. Whenever I end up in an esoteric part of the file system and need to make a quick note of the location, I simply bookmark it.

On SlashDot, that eternal source of wisdom or alternatively the geek wheel of suffering, Clueless Moron offered the following gems.

```
mk() { eval ${1:-MKPWD}=\"`pwd`\"; }
rt() { eval cd \"\$${1:-MKPWD}\";pwd; }

# type "mk" (as in "mark") and "rt" (as in "return") to mark
# a directory and later go back to it.
# Or give it a name: do "mk foo", and later on "rt foo"
```

This of course is a per-session mechanism, but noteworthy for its simplicity. I am not sure whether csh-style shells could offer an equivalent.

A feature shared by apparix and cdargs is that adding a bookmark immediately takes effect in all shells. There is no need to source some resource file, as the applications do this everytime they are invoked. It is fast, do not worry.

BUGS

The resource file parsing code thinks that parentheses are special. Also records are currently separated by commas. Accordingly, apparix will hitch if a path name contains a parenthesis or a comma.

AUTHOR

Stijn van Dongen.

THANKS

Stefan Kamphausen wrote **cdargs**, the inspiration for apparix.

Sitaram Chamarty fixed up some of the existing bash code, and added the tab completion part (basing this on similar code in cdargs). He does not garantuee predictable or even pretty results if there are spaces in the directory names which you attempt to complete. **AUTHOR** would like to submit that spaces in path names are evil, and that the completion code seems to work in their evil presence anyway. Just *don't put commas* in path names.

The autotooled build environment was modified from a template written by Joost van Baal.

HISTORY

Apparix was created to optimize a scenario that *cdargs* does not support very well, namely where the mark (called *needle* in cdargs) is always known. As an additional feature apparix supports CDPATH-style behaviour and subdirectory specification. In other respects apparix is a much simpler application. **cdargs** offers menu-based navigation of the file system and the bookmark list, which apparix does not.