Replacing man with info

GNU `info` is lightyears ahead of `man` in terms of features, with *sub-pages*, *clickable links*, *topic-spanning search*, *clean html- and latex-export* and *efficient interactive navigation*.

But man pages are still the de-facto standard for getting quick information on a GNU/Linux system.

This guide intends to help you change that for your system. It needs `GNU texinfo >= 6.1`.

> **Update**: If you prefer vi-keys, adjust the function below to call `info --vi-keys` instead of plain `info`. You could then call that function `iv`

Contents

1 Advantages of man-pages over pristine info 1

2 Fixing GNU info with a simple bash function 2

3 Examples 4
- 3.1 First check: Getting info on info: . . . 4
- 3.2 Second check: Some random GNU command . . . 4
- 3.3 Utility which also exists as libc function . . . 5
- 3.4 Something which only has a man-page . . . 6
- 3.5 A request for a man page section . . . 6
- 3.6 Something unknown . . . 6

4 Summary 7

1 Advantages of man-pages over pristine info

I see strong reasons for sticking to man pages instead of info: man pages provide what most people need right away (how to use this?) and they fail fast if the topic is not available.

Their advanced features are mostly hidden away (i.e. checking the Linux programmers manual instead of checking installed programs `man 2 stat` vs. `man stat`).

Different from that, the default error state of info is to show you all the other info nodes in which you are really not interested at the moment. And `man basename` gives you the commandline invocation of the basename utility, while `info basename` gives you libc "5.8 Finding Tokens in a String".

Also man is fast. And works on most terminals, while info fails at `dumb` ones.

In short: man does what most users need right now, and if it can't do that, it simply fails, so the user can try something else. That's a huge UI advantage, but not due to an inherent limitation of GNU info. GNU Info can do the same, and even defer to man pages for stuff for which there is no info document. It just does not provide that conveniently by default.

2 Fixing GNU info with a simple bash function

GNU Info can provide the same useful interface as man. So let's make it do that.

To keep all flexibility without needing to adjust the PATH, let's make a bash function. That function can go into `~/.bashrc`, or `/etc/bash/bashrc`.[1] I chose the latter, because it provides the function for all accounts on the system and keeps it separate from the general setup.

The function will be called `i`: To get information about any thing, just call `i thing`.

Let's implement that:

```
function i()
{
    INFOVERSIONLINE=$(info --version | head -n 1)
    INFOVERSION="${INFOVERSIONLINE##* }"
    INFOGT5=$(if test ${INFOVERSION%%.*} -gt 5; then echo true; else echo false; fi)
    # start with special cases which are quick to check for
    if test $# -lt 1; then
        # show info help notice
        info --help
    elif test $# -gt 1 && ! echo $1 | grep -q "[0-9]"; then
        # user sent complex request, but not with a section command. Just use info
        info "$@"
    elif test $# -gt 1 && echo $1 | grep -q "[0-9]"; then
        # user sent request for a section from the man pages, we must defer to man
        man "$@"
```

[1] Or it can go into `/etc/bash/bashrc.d/info.sh` (if you have a bashrc directory). That is the cleanest option.

```
elif test x"$1" = x"info"; then
    # for old versions of info, calling info --usage info fails to
    # provide info about calling info
    if test x"$INFOGT5" = x"true"; then
        info --usage info
    else
        info --usage -f info-stnd
    fi
else
    # start with a fast but incomplete info lookup
    INFOPAGELOCATION="$(info --all -w ./"$@" | head -n 1)"
    INFOPAGELOCATION_PAGENAME="$(info --all -w "$1".info | head -n 1)"
    INFOPAGELOCATION_COREUTILS="$(info -w coreutils -n "$@")"
    # check for usage from fast info, if that fails check man and
    # if that also fails, just get the regular info page.
    if test x"${INFOPAGELOCATION}" = x"*manpages*" || test x"${INFOPAGELOCATION}"
       info --usage "$@"; # use info to read the known page, man or info
    elif test x"${INFOPAGELOCATION_COREUTILS}" != "x" && info -f "${INFOPAGELOCAT
        # coreutils utility
        info -f "${INFOPAGELOCATION_COREUTILS}" -n "$@"
    elif test x"${INFOPAGELOCATION}" = x"" && test x"${INFOPAGELOCATION_PAGENAME}
       # unknown to quick search, try slow search or defer to man.
       # TODO: it would be nice if I could avoid this double search.
       if test x"$(info -w "$@")" = x"*manpages*"; then
           info "$@"
       else
           # defer to man, on error search for alternatives
           man "$@" || (echo nothing found, searching info ... && \
                        while echo $1 | grep -q '^[0-9]$'; do shift; done && \
                        info -k "$@" && false)
       fi
    elif test x"${INFOPAGELOCATION_PAGENAME}" != x""; then
         # search for alternatives (but avoid numbers)
       info --usage -f "${INFOPAGELOCATION_PAGENAME}" 2>/dev/null || man "$@" ||\
         (echo searching info &&\
          while echo $1 | grep -q '^[0-9]$'; do shift; done && \
          info -k "$@" && false)
    else # try to get usage instructions, then try man, then
         # search for alternatives (but avoid numbers)
       info --usage -f "${INFOPAGELOCATION}" 2>/dev/null || man "$@" ||\
         (echo searching info &&\
          while echo $1 | grep -q '^[0-9]$'; do shift; done && \
          info -k "$@" && false)
    fi
```

```
        # ensure that unsuccessful requests report an error status
        INFORETURNVALUE=$?
        unset INFOPAGELOCATION
        unset INFOPAGELOCATION_COREUTILS
        if test ${INFORETURNVALUE} -eq 0; then
            unset INFORETURNVALUE
            return 0
        else
            unset INFORETURNVALUE
            return 1
        fi
    fi
}
```

3 Examples

Let's see what that gives us.

These examples are evaluated on export, so what you see here is the exact result at the time this page was generated.

3.1 First check: Getting info on info:

```
{{{fun}}}
i info | head
echo ...
```

```
File: info-stnd.info,  Node: Invoking Info,  Next: Variables,  Prev: Miscellaneous Co

12 Invoking Info
****************

GNU Info accepts several options to control the initial node or nodes
being viewed, and to specify which directories to search for Info files.
Here is a template showing an invocation of GNU Info from the shell:

     info [OPTION...] [MANUAL] [MENU-OR-INDEX-ITEM...]
...
```

3.2 Second check: Some random GNU command

```
{{{fun}}}
i grep | head | sed 's/^[\[[0-9]*m//g' # stripping simple colors
```

```
echo ...
```

```
File: grep.info,  Node: Invoking,  Next: Regular Expressions,  Prev: Introduction,  U

2 Invoking ‘grep’
*****************

The general synopsis of the ‘grep’ command line is

     grep [OPTION...] [PATTERNS] [FILE...]

There can be zero or more OPTION arguments, and zero or more FILE
...
```

Note: If there's a menu at the bottom, you can jump right to it's entries by hitting the `m` *key.*

3.3 Utility which also exists as libc function

Checking for `i stat` gives us the stat command:

```
{{{fun}}}
i stat | head
```

```
File: coreutils.info,  Node: stat invocation,  Next: sync invocation,  Prev: du invoc

14.3 ‘stat’: Report file or file system status
==============================================

‘stat’ displays information about the specified file(s).  Synopsis:

     stat [OPTION]... [FILE]...

   With no option, ‘stat’ reports all information about the given files.
```

...while checking for `i libc stat` gives us the libc function:

```
{{{fun}}}
i libc stat | head
```

```
File: libc.info,  Node: Reading Attributes,  Next: Testing File Type,  Prev: Attribut

14.10.2 Reading the Attributes of a File
----------------------------------------

To examine the attributes of files, use the functions ‘stat’, ‘fstat’
and ‘lstat’.  They return the attribute information in a ‘struct stat’
```

```
object.  All three functions are declared in the header file
‘sys/stat.h’.
```

3.4 Something which only has a man-page

`i man` cleanly calls `info man`.

```
{{{fun}}}
i man | head | sed "s,\x1B\[[0-9;]*[a-zA-Z],,g" # stripping colors
```

No output here, because only the interactive terminal shows the output.

3.5 A request for a man page section

`i 2 stat` cleanly defers to `man 2 stat`

```
{{{fun}}}
i 2 stat | head | sed "s,\x1B\[[0-9;]*[a-zA-Z],,g" # stripping colors
```

No output here, because the export system does not have libc man pages installed.

3.6 Something unknown

In case there is no info directly available, do a keyword search and propose sources.

```
{{{fun}}}
i em | head
echo ...
```

```
nothing found, searching info ...
"(bash)Bash Variables" -- BASH_REMATCH
"(bash)Readline Init File Syntax" -- completion-query-items
"(bash)Bash Variables" -- EMACS
"(bash)Readline Init File Syntax" -- emacs-mode-string
"(bash)Bash Variables" -- INSIDE_EMACS
"(bash)Commands For History" -- non-incremental-forward-search-history (M-n)
"(bash)Commands For History" -- non-incremental-reverse-search-history (M-p)
"(coreutils)rm invocation" -- -, removing files beginning with
"(coreutils)cut invocation" -- --complement
...
```

4 Summary

`i thing` gives you info on some thing. It makes using info just as convenient as using man.

Its usage even beats man in convenience, since it defers to man if needed, offers alternatives and provides named categories instead of having to remember the handbook numbers to find the right function.

And as developer you can use texinfo to provide high quality documentation in many formats. You can even include a comprehensive tutorial in your documentation while still enabling your users to quickly reach the information they need.

We had this all along, except for a few nasty roadblocks. Here I did my best to eliminate these roadblocks.