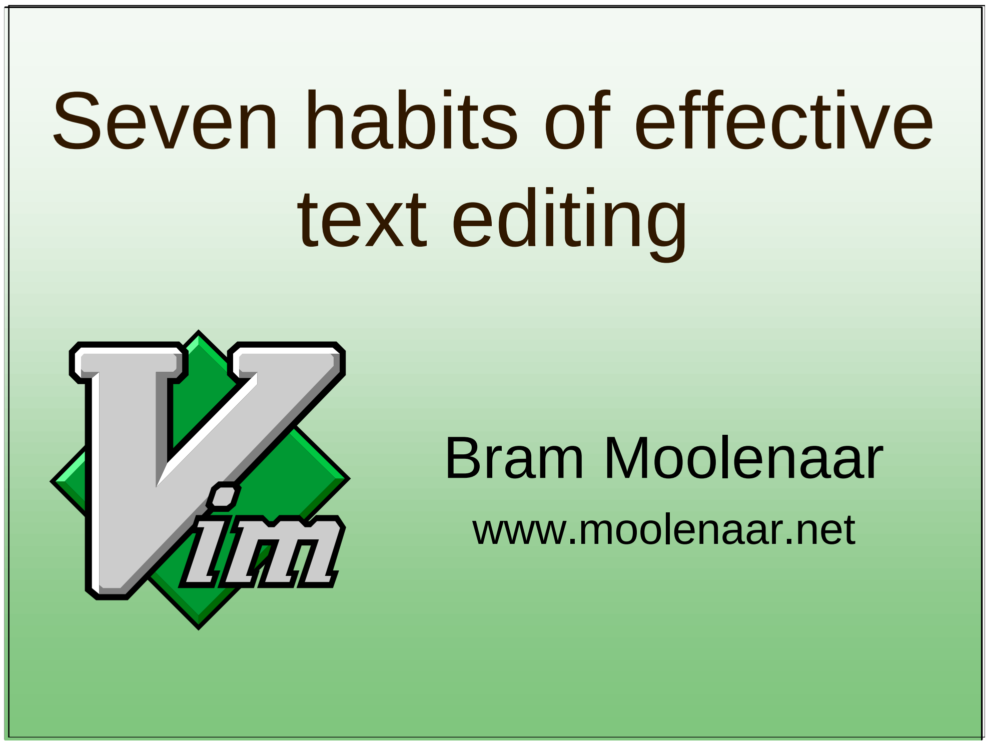


Seven habits of effective text editing

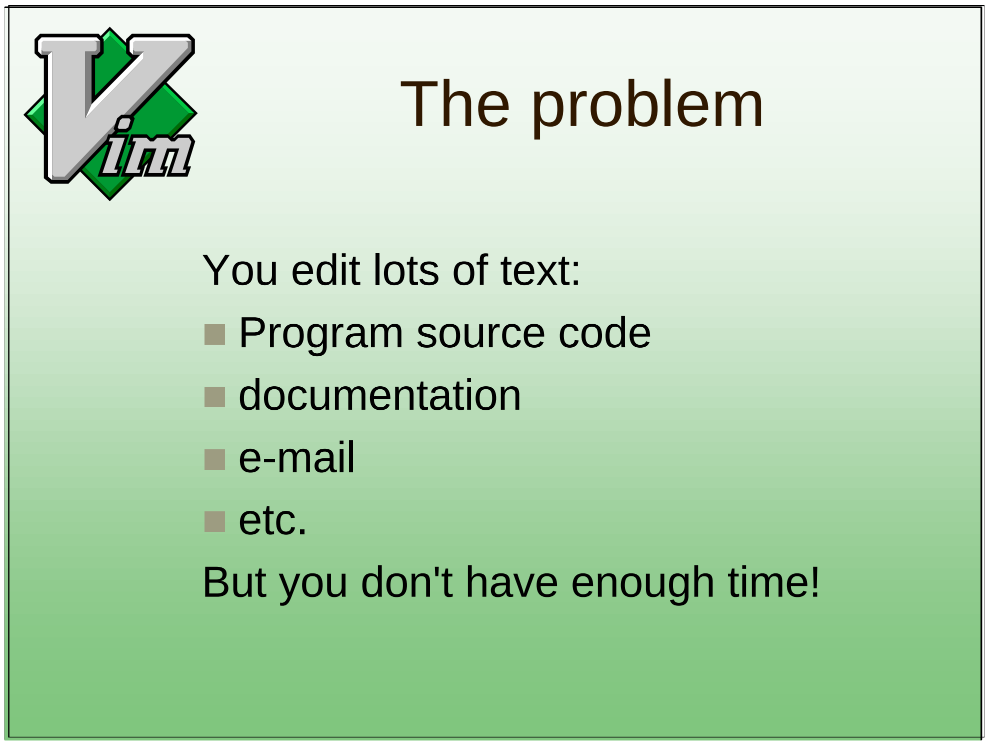
Bram Moolenaar

www.moolenaar.net

Presentation given by Bram Moolenaar at the NLUUG conference, November 9 2000.

I can do this presentation in Dutch or English. If there is someone who doesn't understand Dutch I'll use English.

There will be time to ask questions at the end of the presentation.



The problem

You edit lots of text:

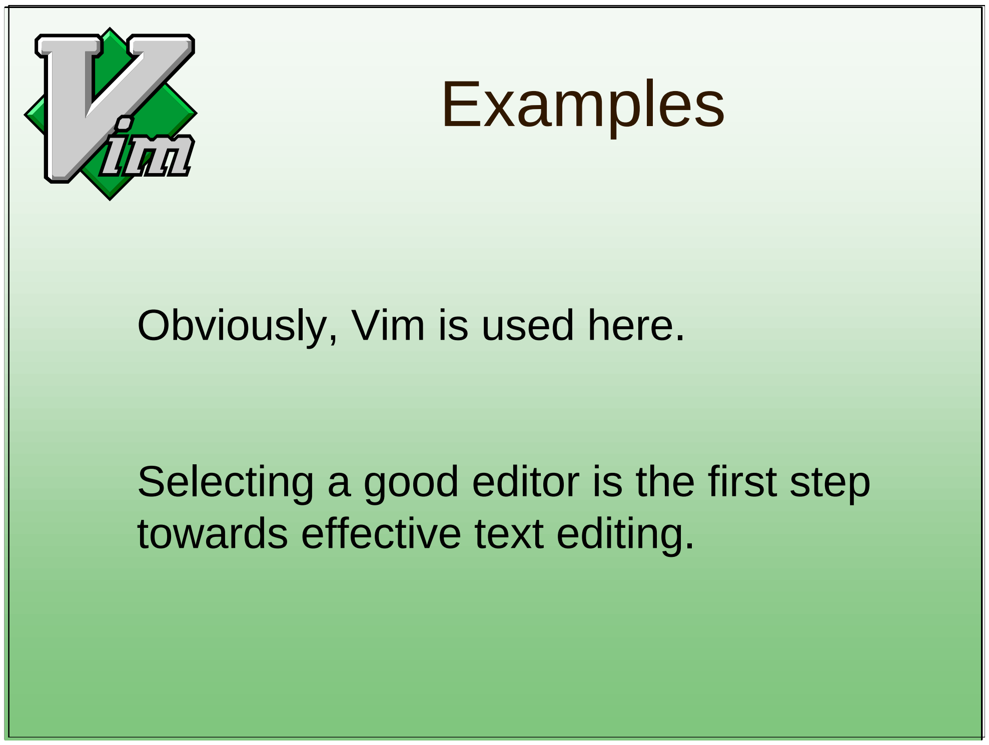
- Program source code
- documentation
- e-mail
- etc.

But you don't have enough time!

Editing text takes a lot of our time. Looking at myself, I spend more than half my day reading and changing various kinds of text.

This presentation is about how to get more work done in less time.

Don’t listen when you are paid by the hour. :-)



Examples

Obviously, Vim is used here.

Selecting a good editor is the first step towards effective text editing.

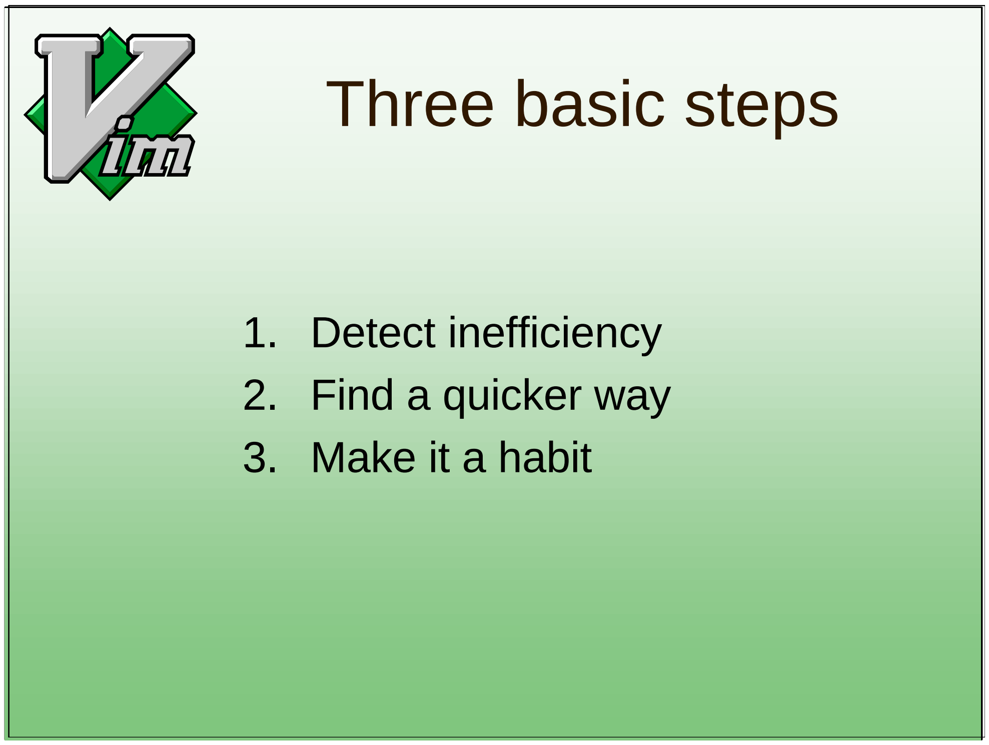
Some people use Notepad and never get past it. Those people can learn a lot today.

You can spend lots of time on evaluating different editors. I use two rules:

- If you are already using an editor and it works very well for you, don’t waste time by learning to use another one. However, after this presentation you might wonder if your editor is really good enough.
- Otherwise use Vim. You won’t be disappointed.

I’m not going into the discussion which editor is best for you. That would take too much time.

Question: Who of you have never used Vim? Who is using Vim every day?



Three basic steps

1. Detect inefficiency
2. Find a quicker way
3. Make it a habit

Keep en eye out for actions that you repeat and/or spend a lot of time on. Lean back and evaluate what you have been doing the past hour. If you have more than a bit of work to do, try to find a pattern in it that repeats itself.

Example: You have a program with several files and have to rename a function that's used in many places. If you have to type the name many times you are inefficient.

Any powerful editor provides you with commands to do your work fast. You just have to find them. If the editor doesn't offer an appropriate command, you can use its script language. Or perhaps you can use an external program.

You have to learn new commands until you use them automatically.

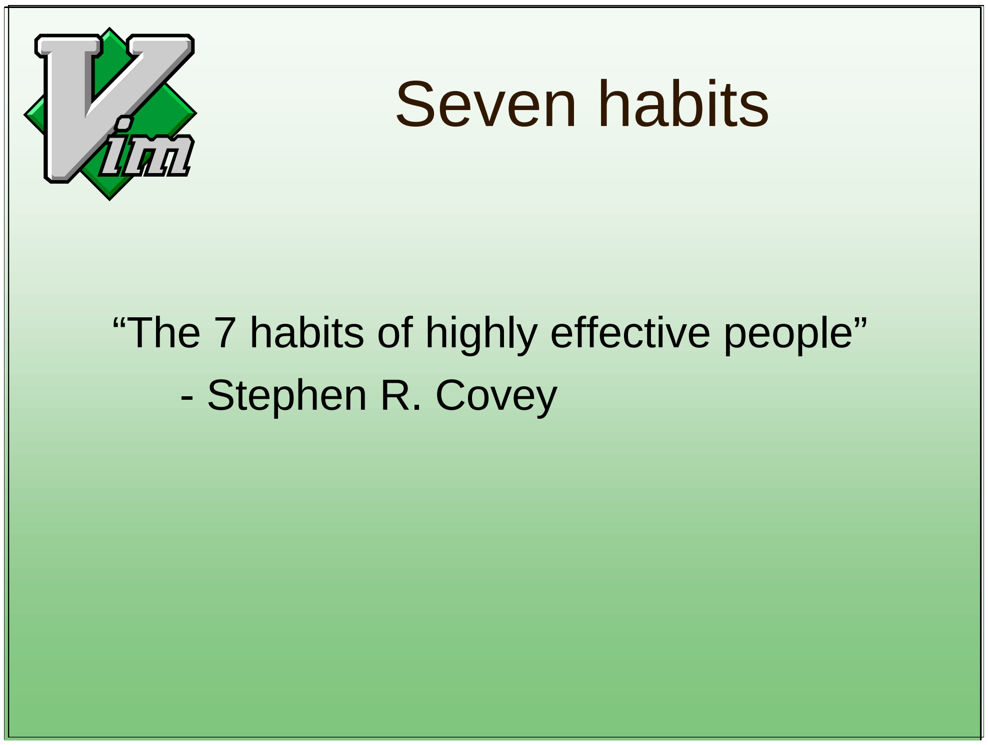
Summary:

1. See the problem

2. find a solution

3. use it

That sounds easy, doesn't it? And so it is, you just have to do it.



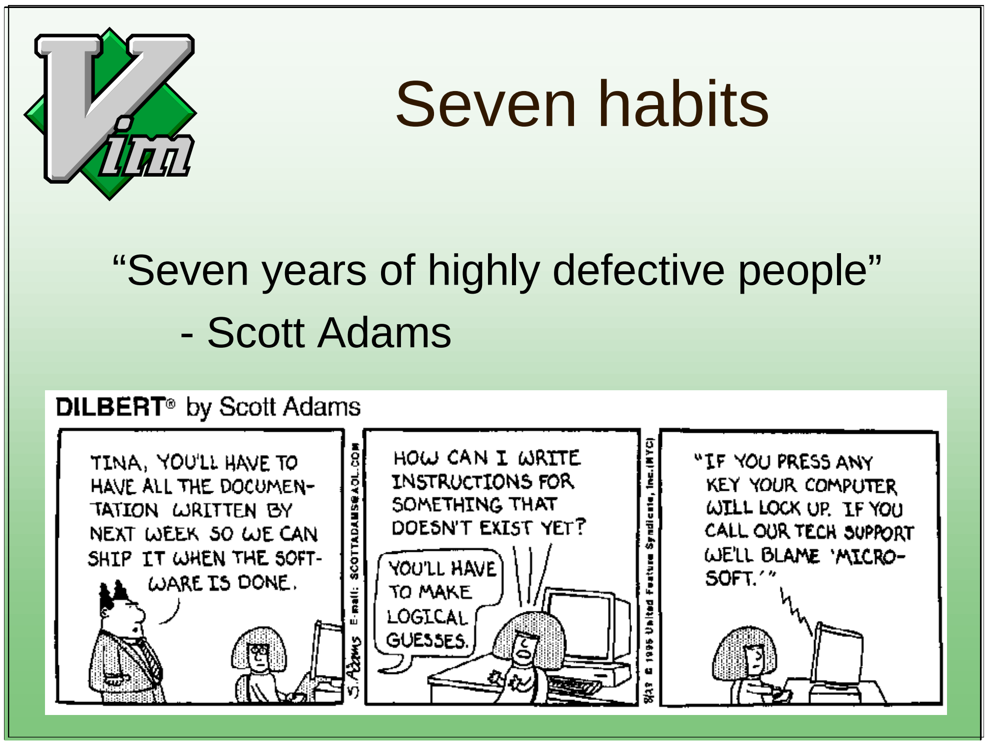
Seven habits

"The 7 habits of highly effective people"
- Stephen R. Covey

I will give seven examples. Why seven? To match the title. The title is inspired by the book from Stephen Covey.

The Seven habits book is a very good one. I can recommend it to anyone who wants to improve his live. And who doesn't?

Although, the presentation might as well be based on Dilbert….



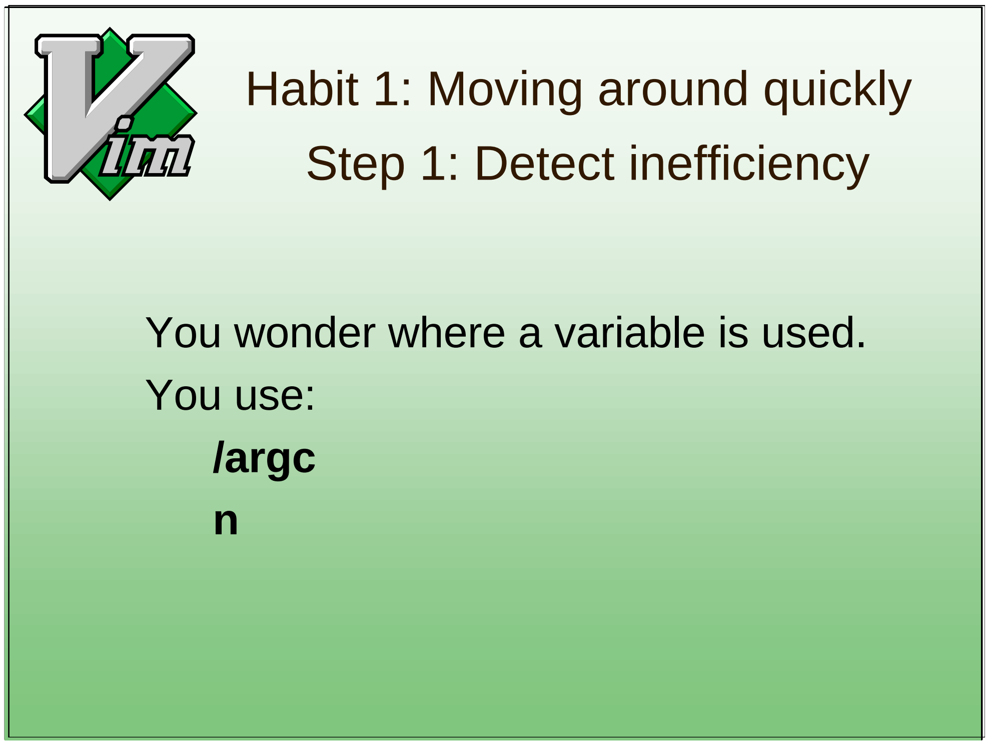
Seven habits

"Seven years of highly defective people"
- Scott Adams

DILBERT® by Scott Adams
TINA, YOU'LL HAVE TO HAVE ALL THE DOCUMENTATION WRITTEN BY NEXT WEEK SO WE CAN SHIP IT WHEN THE SOFTWARE IS DONE.
HOW CAN I WRITE INSTRUCTIONS FOR SOMETHING THAT DOESN'T EXIST YET?
YOU'LL HAVE TO MAKE LOGICAL GUESSES.
"IF YOU PRESS ANY KEY YOUR COMPUTER WILL LOCK UP. IF YOU CALL OUR TECH SUPPORT WE'LL BLAME 'MICROSOFT.'"
E-mail: SCOTTADAMS@AOL.COM
© 1995 United Feature Syndicate, Inc.(NYC)

The Dilbert book helps you to enjoy life. Read it when you take a break.

These two books make a great combination. See **http://www.vim.org/iccf/click1.html** for more information (click on "recommended books and CDs").



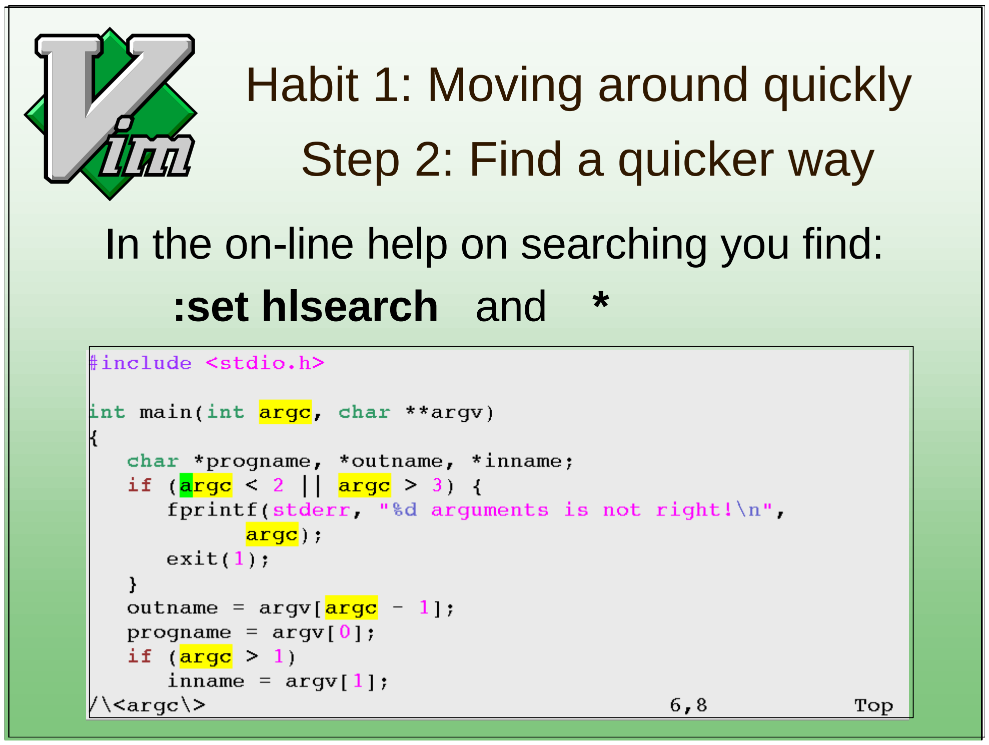
Habit 1: Moving around quickly
Step 1: Detect inefficiency

You wonder where a variable is used.

You use:

```
/argc
n
```

While editing a program you often have to check where a variable is set and where it’s used. Currently you use the search command to find each location.



Habit 1: Moving around quickly

Step 2: Find a quicker way

In the on-line help on searching you find:

:set hlsearch   and   *****

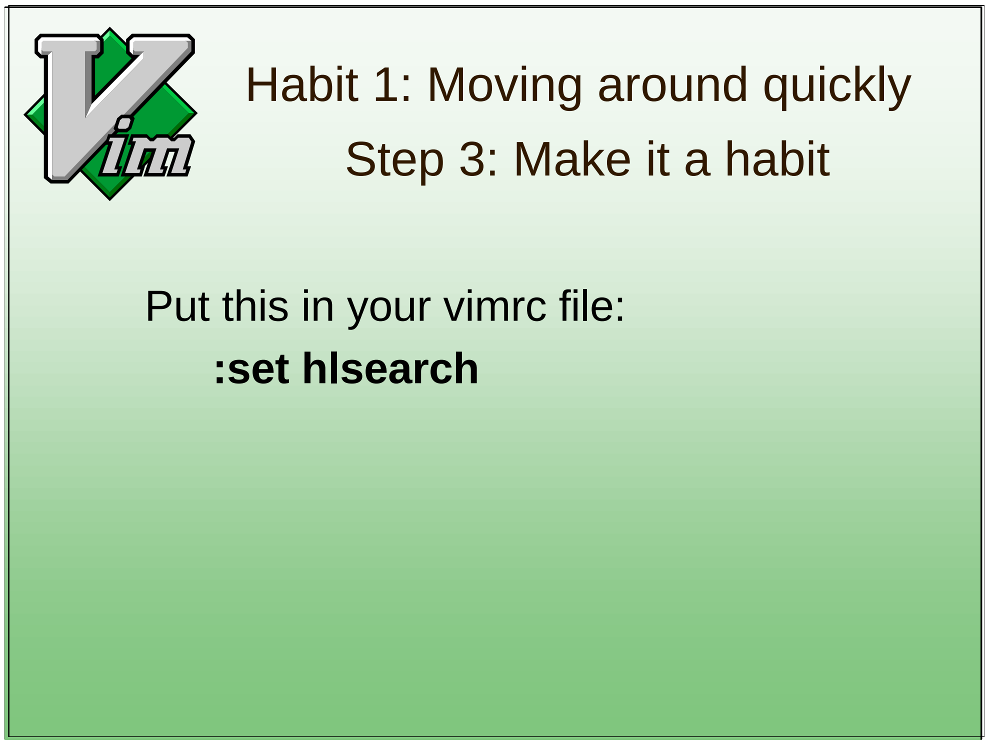
```
#include <stdio.h>

int main(int argc, char **argv)
{
   char *progname, *outname, *inname;
   if (argc < 2 || argc > 3) {
      fprintf(stderr, "%d arguments is not right!\n",
            argc);
      exit(1);
   }
   outname = argv[argc - 1];
   progname = argv[0];
   if (argc > 1)
      inname = argv[1];
/\<argc\>                                   6,8           Top
```

You look in the on-line help for searching commands. You find references to two items that appear to be useful:

The hlsearch option shows all matches with a pattern. You don’t have to find each match, you can see them right away.

The * command searches for the word under the cursor. You don’t have to type the word.



Habit 1: Moving around quickly
Step 3: Make it a habit

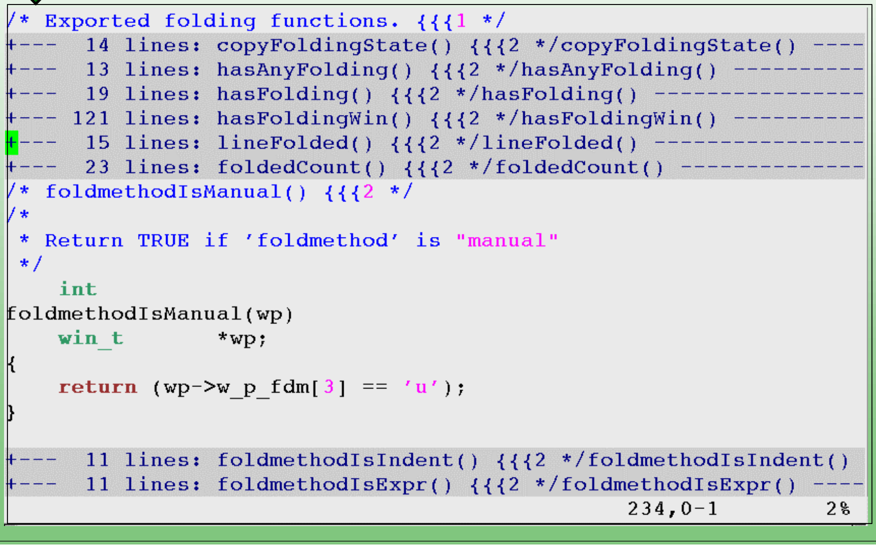
Put this in your vimrc file:

```
:set hlsearch
```

You now start using the “*” command.

Every editor offers many ways to move around. It’s not a bad idea to read the documentation to find useful commands.

Habit 1: Moving around quickly (folding in Vim 6.0)

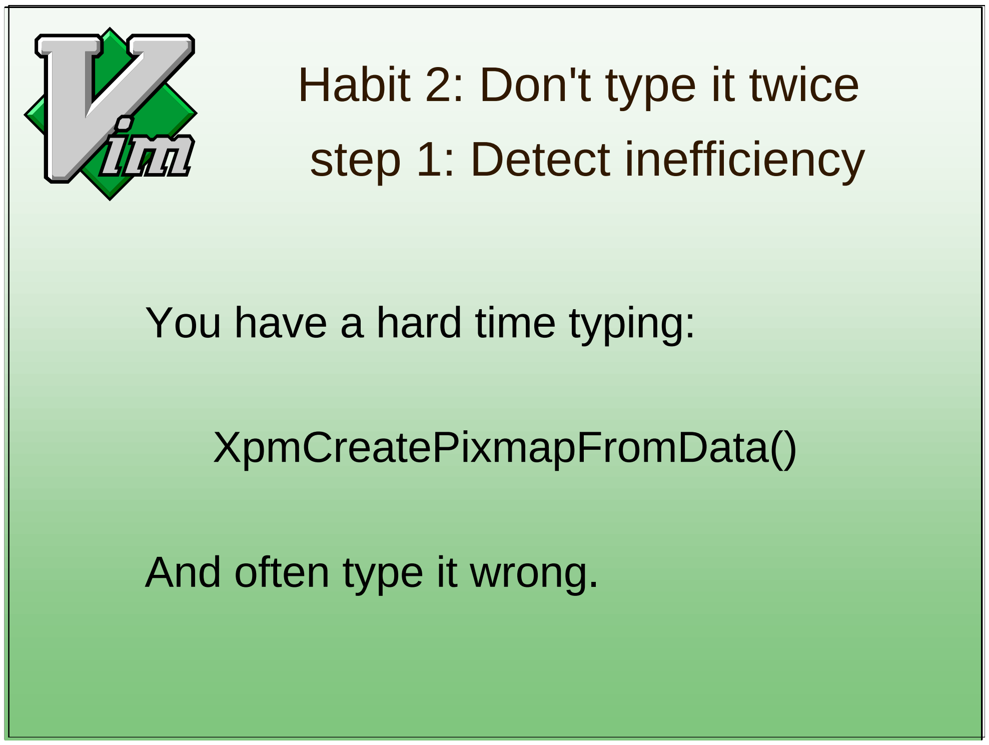


```
/* Exported folding functions. {{{1 */
+---  14 lines: copyFoldingState() {{{2 */copyFoldingState() ----
+---  13 lines: hasAnyFolding() {{{2 */hasAnyFolding() ----------
+---  19 lines: hasFolding() {{{2 */hasFolding() ----------------
+--- 121 lines: hasFoldingWin() {{{2 */hasFoldingWin() ----------
+---  15 lines: lineFolded() {{{2 */lineFolded() ----------------
+---  23 lines: foldedCount() {{{2 */foldedCount() --------------
/* foldmethodIsManual() {{{2 */
/*
 * Return TRUE if 'foldmethod' is "manual"
 */
    int
foldmethodIsManual(wp)
    win_t       *wp;
{
    return (wp->w_p_fdm[3] == 'u');
}

+---  11 lines: foldmethodIsIndent() {{{2 */foldmethodIsIndent()
+---  11 lines: foldmethodIsExpr() {{{2 */foldmethodIsExpr() ----
                                               234,0-1        2%
```

The folds contain a block of lines. In this example each fold contains a function.

Closing all the folds makes it easy to locate a function, quickly move to it and open the fold. That’s a clever way to move around quickly.



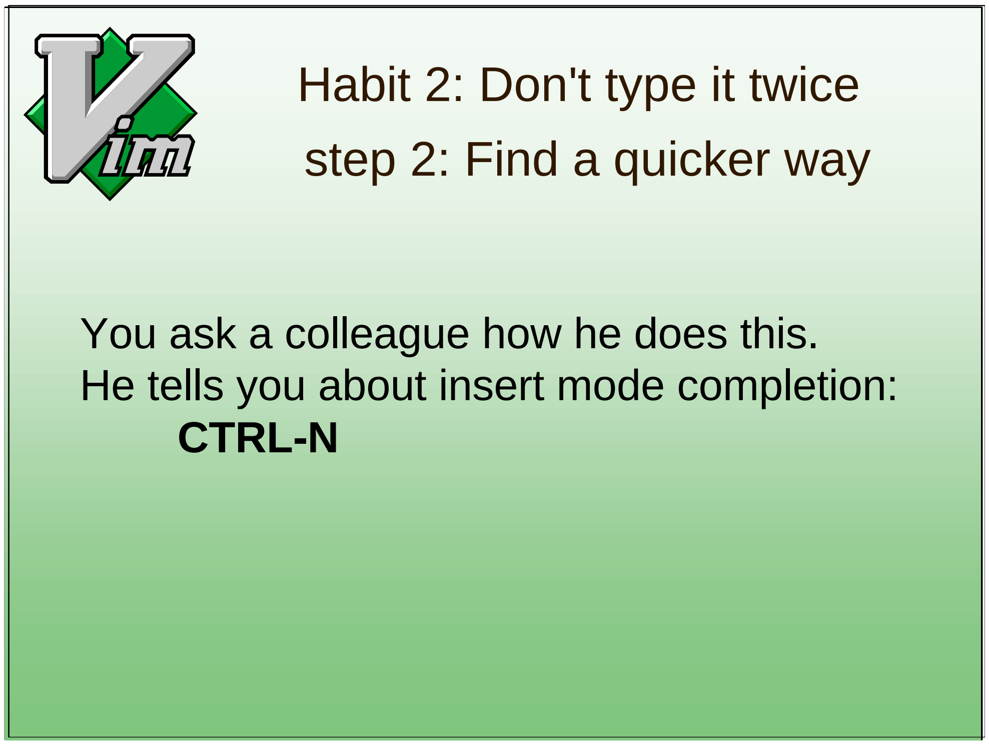
Habit 2: Don't type it twice
step 1: Detect inefficiency

You have a hard time typing:

```
XpmCreatePixmapFromData()
```

And often type it wrong.

Function names can be very difficult to type. It takes a lot of key strokes and it’s easy to make a mistake.



Habit 2: Don't type it twice

step 2: Find a quicker way

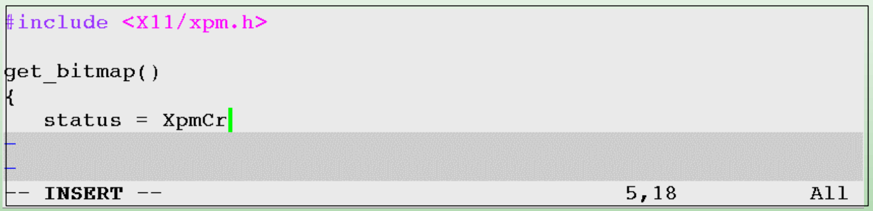
You ask a colleague how he does this.
He tells you about insert mode completion:
CTRL-N

This time you ask someone else if he knows a quicker way. He has run into the same problem before and found a good solution: Insert mode completion.

It's always a good idea to ask someone else, you don't have to invent the solution yourself.

Habit 2: Don't type it twice

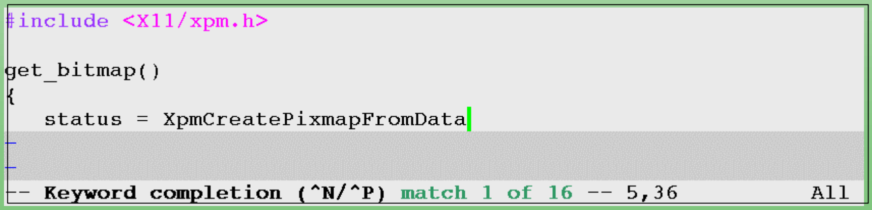
step 3: Make it a habit



```
#include <X11/xpm.h>

get_bitmap()
{
   status = XpmCr
~
~
-- INSERT --                                      5,18           All
```

type **CTRL-N**



```
#include <X11/xpm.h>

get_bitmap()
{
   status = XpmCreatePixmapFromData
~
~
-- Keyword completion (^N/^P) match 1 of 16 -- 5,36           All
```

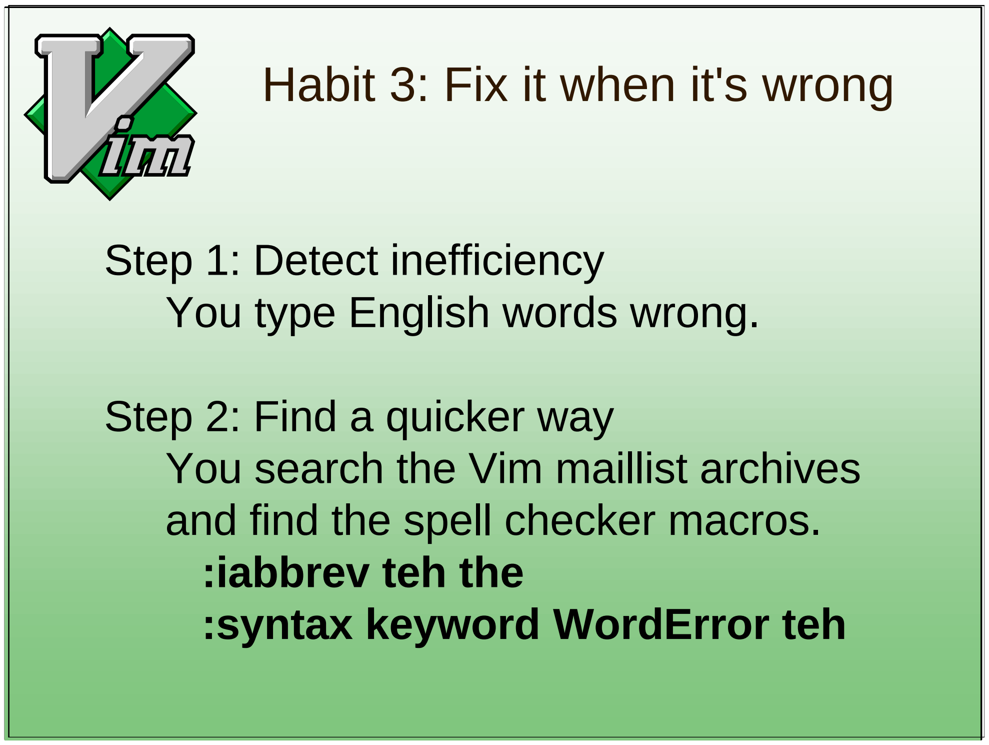
You start using it and find out what you need to type to complete it quickly.

If you don’t type enough you get too many matches. Type CTRL-P to go back to the original word, type an extra letter and hit CTRL-N again.

If you don’t get a match you forgot the #include line.

After doing this for a while you hardly type any long words.

It’s also useful when typing names of Indian people in e-mail. You can create a dictionary for specific words you use.



Habit 3: Fix it when it's wrong

Step 1: Detect inefficiency
    You type English words wrong.

Step 2: Find a quicker way
    You search the Vim maillist archives
    and find the spell checker macros.

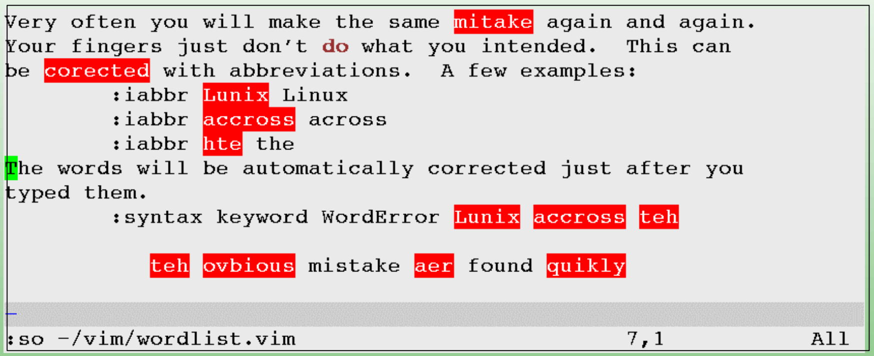
```
:iabbrev teh the
:syntax keyword WordError teh
```

When typing English text you often make mistakes. You have to proofread your text carefully or use a spelling checker afterwards.

This time you look for a solution on the internet. The Vim maillist contains announcements for scripts that people made. You can find an archive at:

http://www.egroups.com/list/vim

Habit 3: Fix it when it's wrong



```
Very often you will make the same mitake again and again.
Your fingers just don't do what you intended.  This can
be corected with abbreviations.  A few examples:
        :iabbr Lunix Linux
        :iabbr accross across
        :iabbr hte the
The words will be automatically corrected just after you
typed them.
        :syntax keyword WordError Lunix accross teh

           teh ovbious mistake aer found quikly

:so ~/vim/wordlist.vim                         7,1           All
```

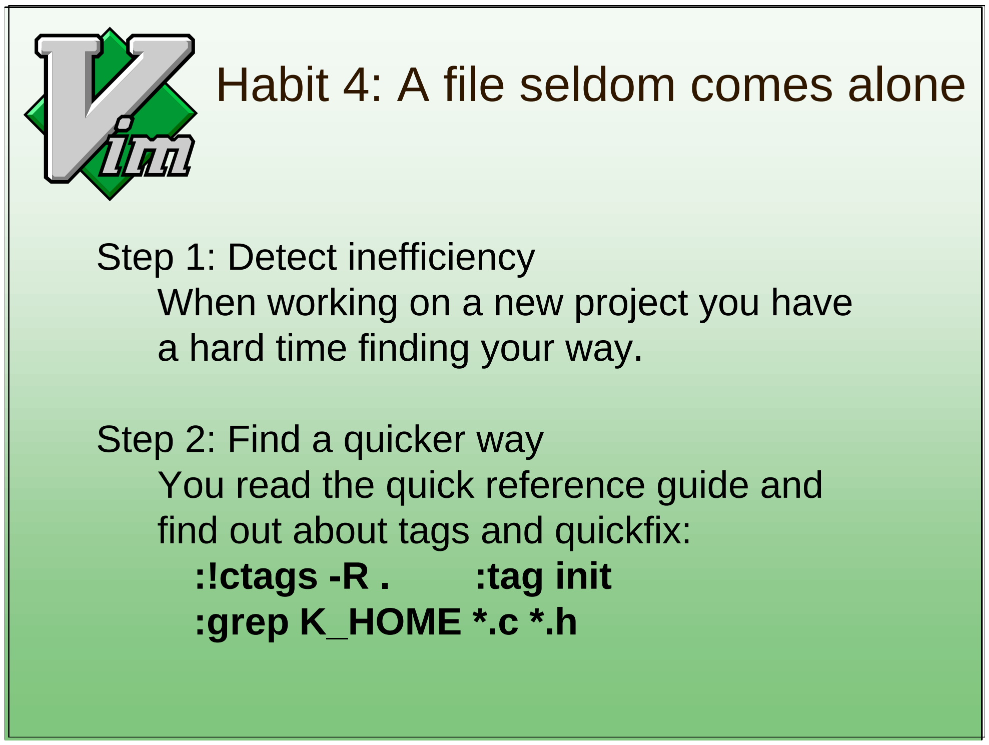
Step 3: make it a habit

Add new words if you see them.

Now you spot typing mistakes when proofreading text. And they are automatically corrected when you type them.

Actually, it was a bit difficult to creaate this example, since the mistakes were automatically corrected when I typed them.

Whenever you spot a wrong word that isn’t detected yet you add it to the dictionary.



Habit 4: A file seldom comes alone

Step 1: Detect inefficiency
When working on a new project you have a hard time finding your way.

Step 2: Find a quicker way
You read the quick reference guide and find out about tags and quickfix:

```
:!ctags -R .       :tag init
:grep K_HOME *.c *.h
```

When starting to look through a new project it's hard to find the relations between where a function or variable is used and where it's defined. Without special tools you waste a lot of time looking around.

The quick reference guide is available from the Vim ftp site:

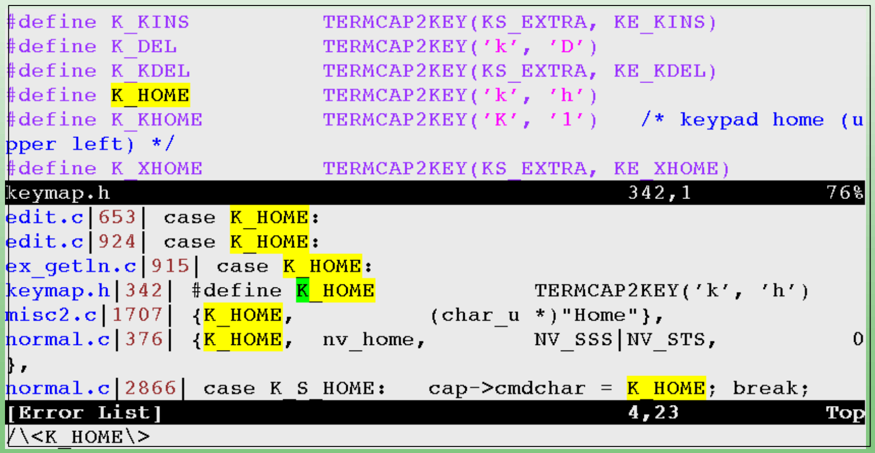
ftp://ftp.vim.org/pub/vim/doc/

A tag file can be used to jump to where an item is defined. Exuberant ctags is recommended http://ctags.sourceforge.net/

Question: who of you hear about using tags for the first time?

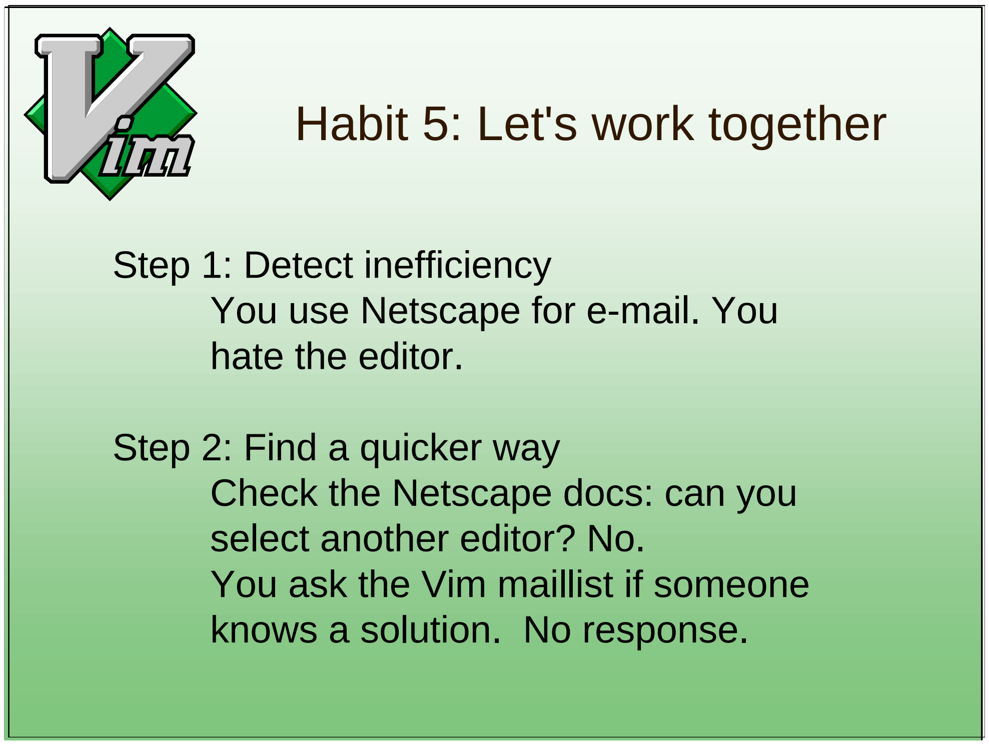
When searching for all places where a variable or function is used, the ":grep" command is useful. You can jump to each next item with ":cn".

Habit 4: A file seldom comes alone
Vim 6.0: quickfix window



```
#define K_KINS          TERMCAP2KEY(KS_EXTRA, KE_KINS)
#define K_DEL           TERMCAP2KEY('k', 'D')
#define K_KDEL          TERMCAP2KEY(KS_EXTRA, KE_KDEL)
#define K_HOME          TERMCAP2KEY('k', 'h')
#define K_KHOME         TERMCAP2KEY('K', '1')   /* keypad home (u
pper left) */
#define K_XHOME         TERMCAP2KEY(KS_EXTRA, KE_XHOME)
keymap.h                                          342,1          76%
edit.c|653|  case K_HOME:
edit.c|924|  case K_HOME:
ex_getln.c|915|  case K_HOME:
keymap.h|342|  #define K_HOME          TERMCAP2KEY('k', 'h')
misc2.c|1707|  {K_HOME,          (char_u *)"Home"},
normal.c|376|  {K_HOME,  nv_home,        NV_SSS|NV_STS,          0
},
normal.c|2866|  case K_S_HOME:   cap->cmdchar = K_HOME; break;
[Error List]                                      4,23          Top
/\<K_HOME\>
```

The quickfix window lists all items from a ":make" or ":grep" command. This gives you an overview and allows quickly jumping to the location you want to see: Hitting <Tab> on a line displays the line with that error in the other window.



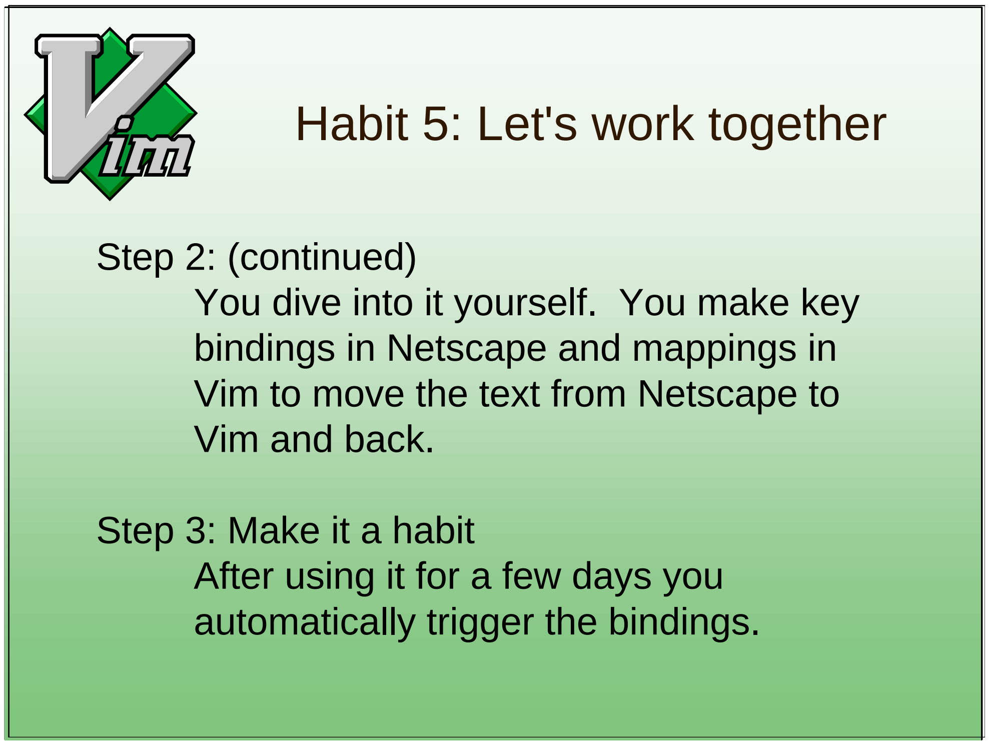
Habit 5: Let's work together

Step 1: Detect inefficiency
You use Netscape for e-mail. You hate the editor.

Step 2: Find a quicker way
Check the Netscape docs: can you select another editor? No.
You ask the Vim maillist if someone knows a solution. No response.

Real power comes from programs working together. Netscape is a good browser but has a bad editor, Vim is a good editor but not a browser. Connecting the two will use the best of both.

Unfortunately, you can’t find an existing solution for this.



Habit 5: Let's work together

Step 2: (continued)

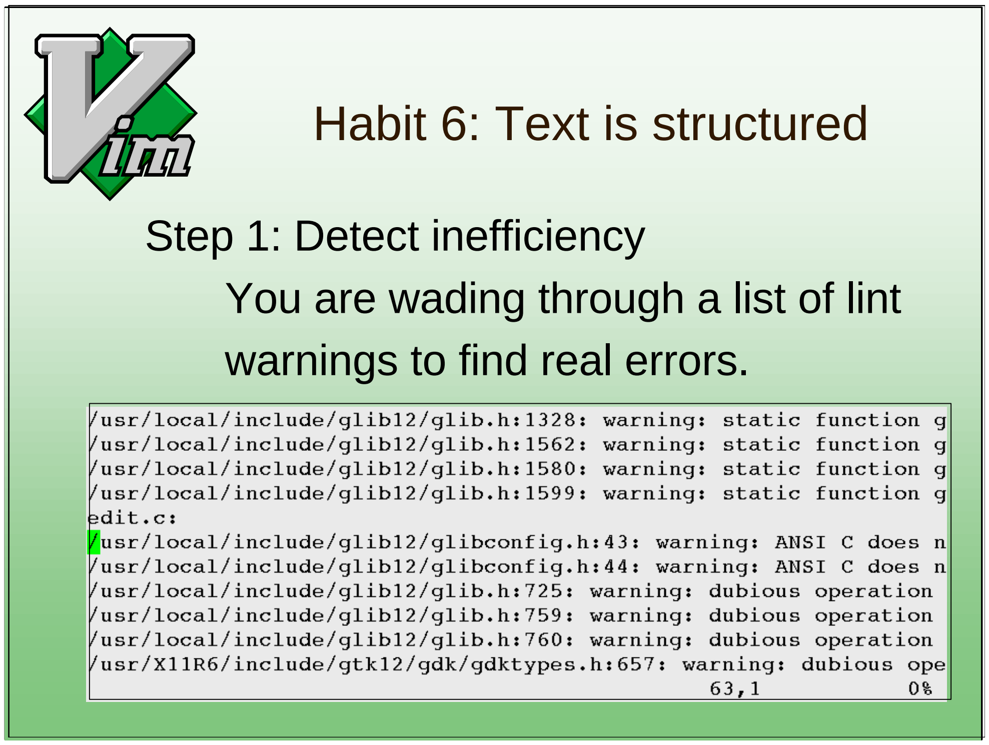
You dive into it yourself. You make key bindings in Netscape and mappings in Vim to move the text from Netscape to Vim and back.

Step 3: Make it a habit

After using it for a few days you automatically trigger the bindings.

So, this is a tough one. But since you really waste a lot of time with the Netscape editor you take the time to find out how to copy the text between Netscape and Vim with a few keystrokes.

This is actually not done yet. If you find the solution, please send it to the Vim maillist, so that others can use it too!



Habit 6: Text is structured

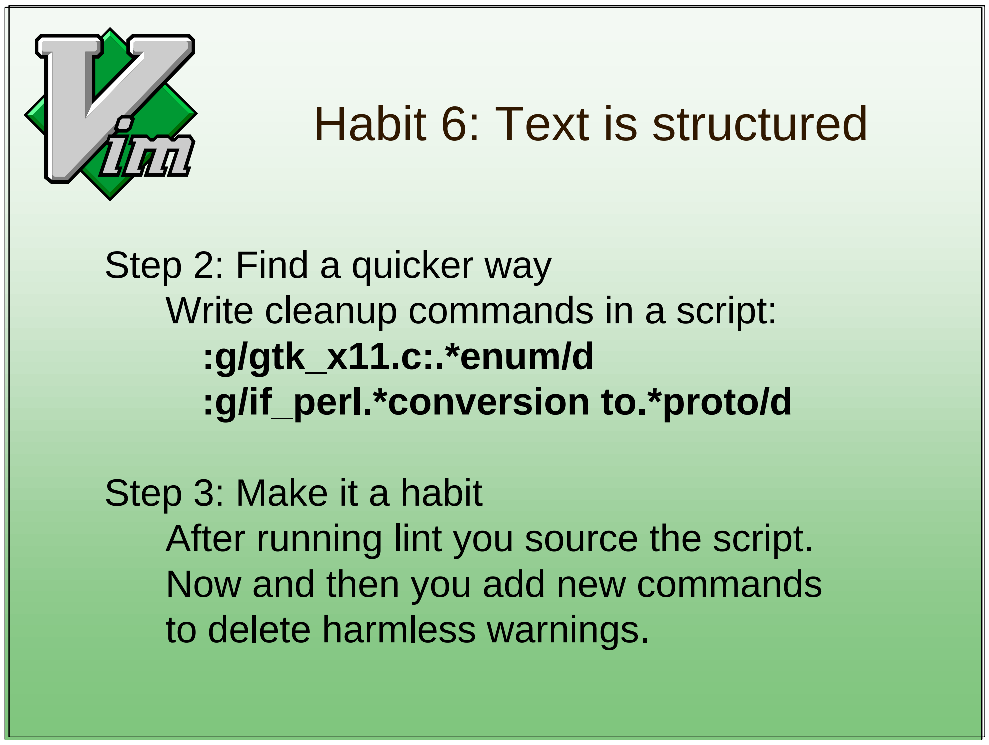
Step 1: Detect inefficiency

You are wading through a list of lint warnings to find real errors.

```
/usr/local/include/glib12/glib.h:1328: warning: static function g
/usr/local/include/glib12/glib.h:1562: warning: static function g
/usr/local/include/glib12/glib.h:1580: warning: static function g
/usr/local/include/glib12/glib.h:1599: warning: static function g
edit.c:
/usr/local/include/glib12/glibconfig.h:43: warning: ANSI C does n
/usr/local/include/glib12/glibconfig.h:44: warning: ANSI C does n
/usr/local/include/glib12/glib.h:725: warning: dubious operation
/usr/local/include/glib12/glib.h:759: warning: dubious operation
/usr/local/include/glib12/glib.h:760: warning: dubious operation
/usr/X11R6/include/gtk12/gdk/gdktypes.h:657: warning: dubious ope
                                               63,1          0%
```

Even though files contain plain text, it is often structured. You can use this to make your editing more productive.

If you have warnings in include files you can’t avoid them. It makes it difficult to find the serious warnings that you need to take care of.



Habit 6: Text is structured

Step 2: Find a quicker way

Write cleanup commands in a script:

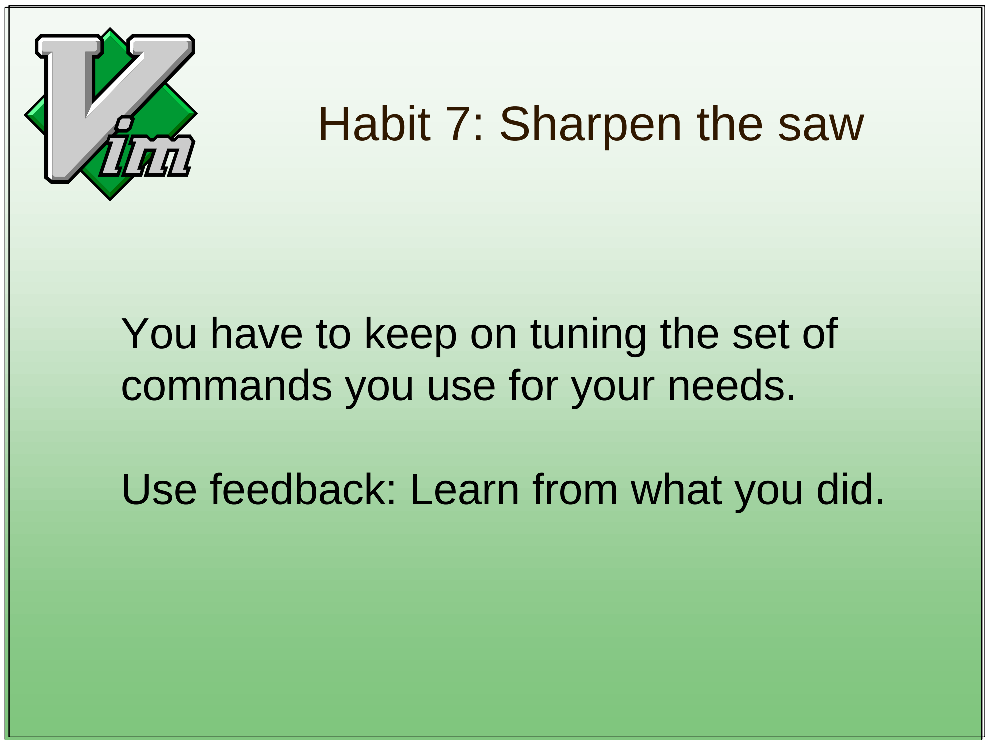
```
:g/gtk_x11.c:.*enum/d
:g/if_perl.*conversion to.*proto/d
```

Step 3: Make it a habit

After running lint you source the script.
Now and then you add new commands
to delete harmless warnings.

This time you decide to use the features the editor offers to extend its functionality. Most editors offer some script language. Vim has Vim script, which uses the same Ex commands that you type interactively.

The trick here is to use the right patterns to only match the lines of harmless warnings. You need to tune this to avoid that serious warnings get deleted, or that harmless warnings clobber the list.



Habit 7: Sharpen the saw

You have to keep on tuning the set of commands you use for your needs.

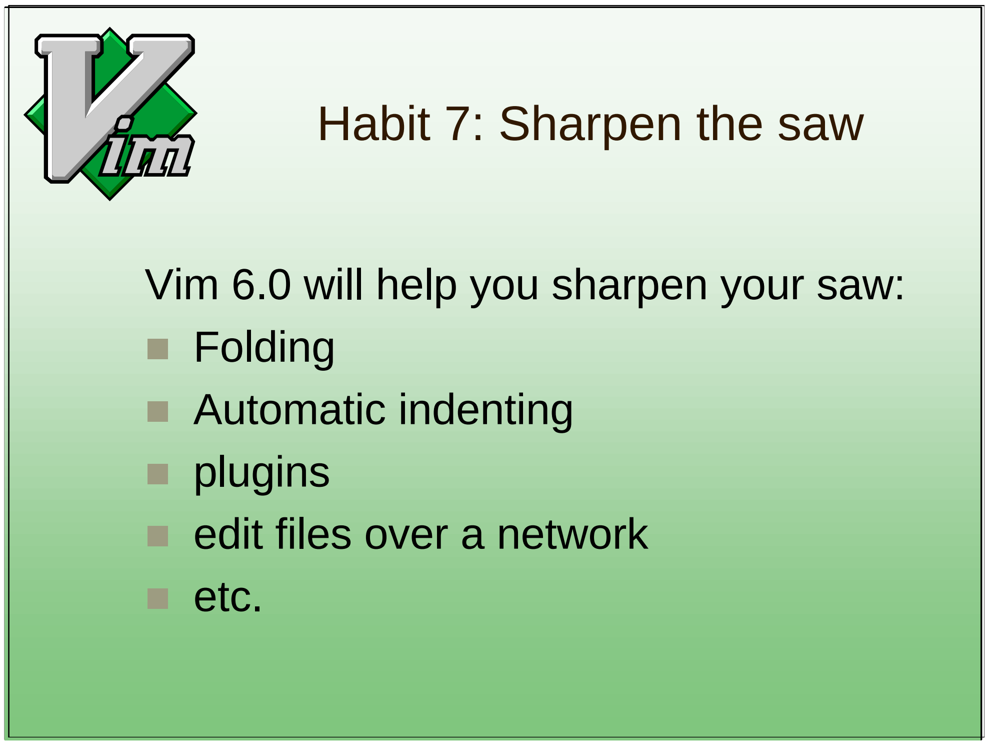
Use feedback: Learn from what you did.

The third step seems to come down to “do it”. You might think that’s obvious and not important. The contrary is true. Only when you have made the task a habit will you be able to work at high speed.

Compare to learning to drive a car. The first few lessons you have to think about how to get it in the next gear. Only after practicing a while you can do it automatically and have time to think about where you are going.

Learning to use an editor is similar, with one important difference: It has many more commands. Too many to learn them all. You need to learn one at a time, when you need it.

On a meta level you should think about what you did with the editor. How much time did you waste on not using the best commands? How much time did you waste on finding a better way? How much time did you win by writing that macro? Use this to know what you need to do in the future.



Habit 7: Sharpen the saw

Vim 6.0 will help you sharpen your saw:

- Folding
- Automatic indenting
- plugins
- edit files over a network
- etc.

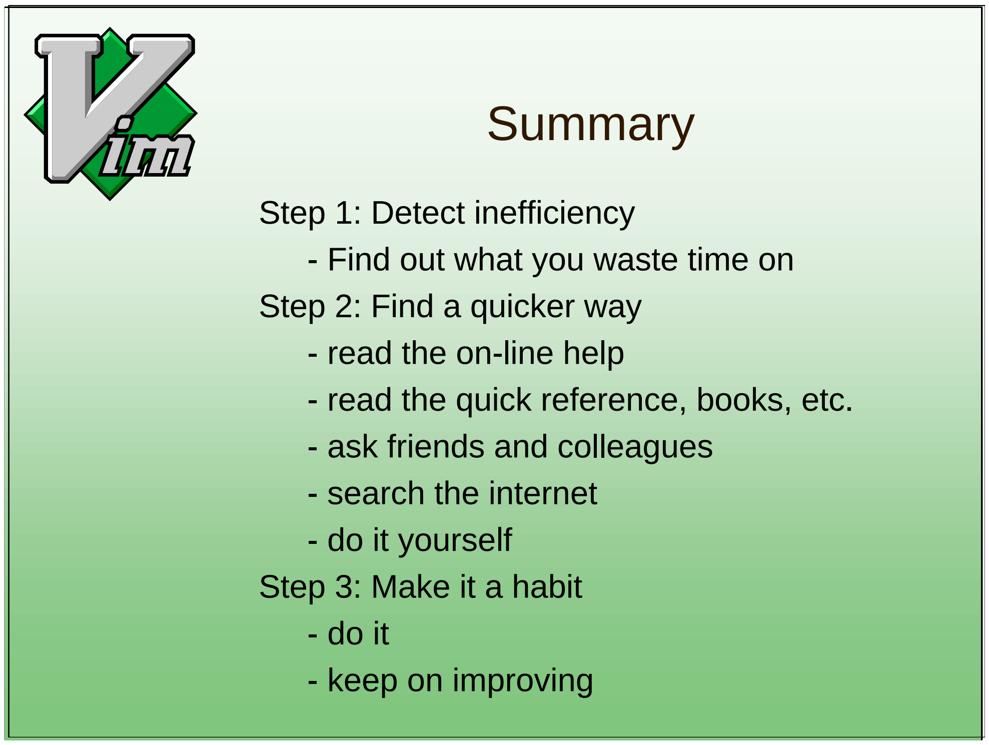
Vim 6.0 will have many more features. All of them have been asked for by users. Thus it should make editing more effective for you too.

Automatic indenting is very flexible. You can define indenting for your specific needs. This avoids manually adjusting the indent.

Plugins make it easy to add functionality and exchange scripts between users.

Editing files over a network saves you the manual commands to make a copy of a file and write it back later. This is actually done by a plugin.

Vim 6.0 is still under development. Hopefully it will be ready early next year.

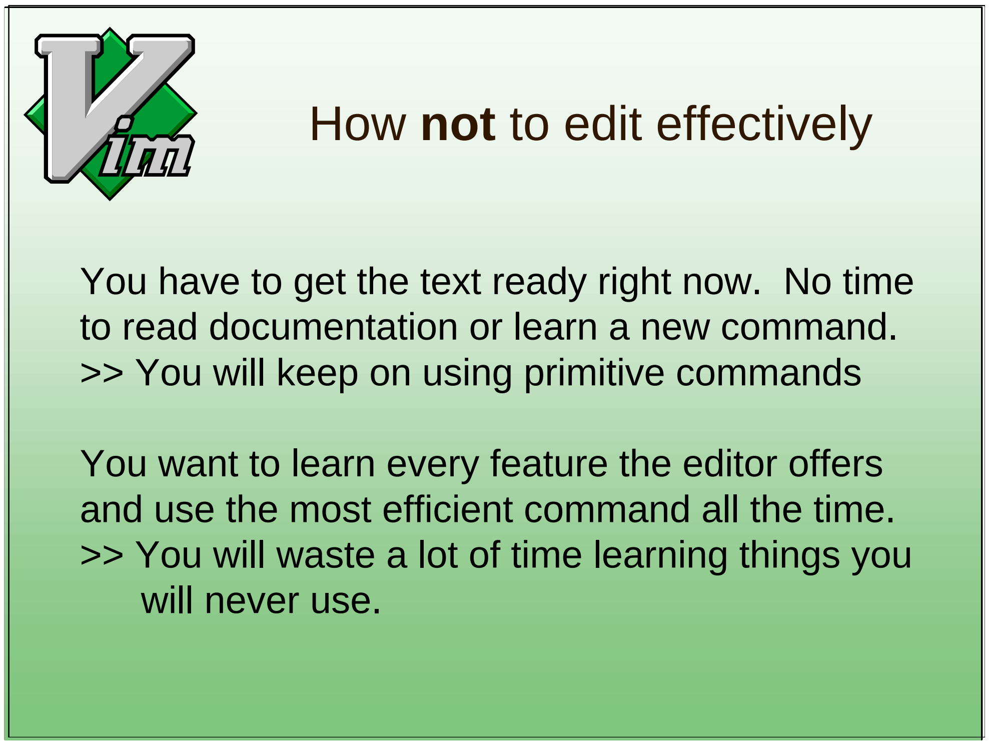


Summary

Step 1: Detect inefficiency
- Find out what you waste time on

Step 2: Find a quicker way
- read the on-line help
- read the quick reference, books, etc.
- ask friends and colleagues
- search the internet
- do it yourself

Step 3: Make it a habit
- do it
- keep on improving



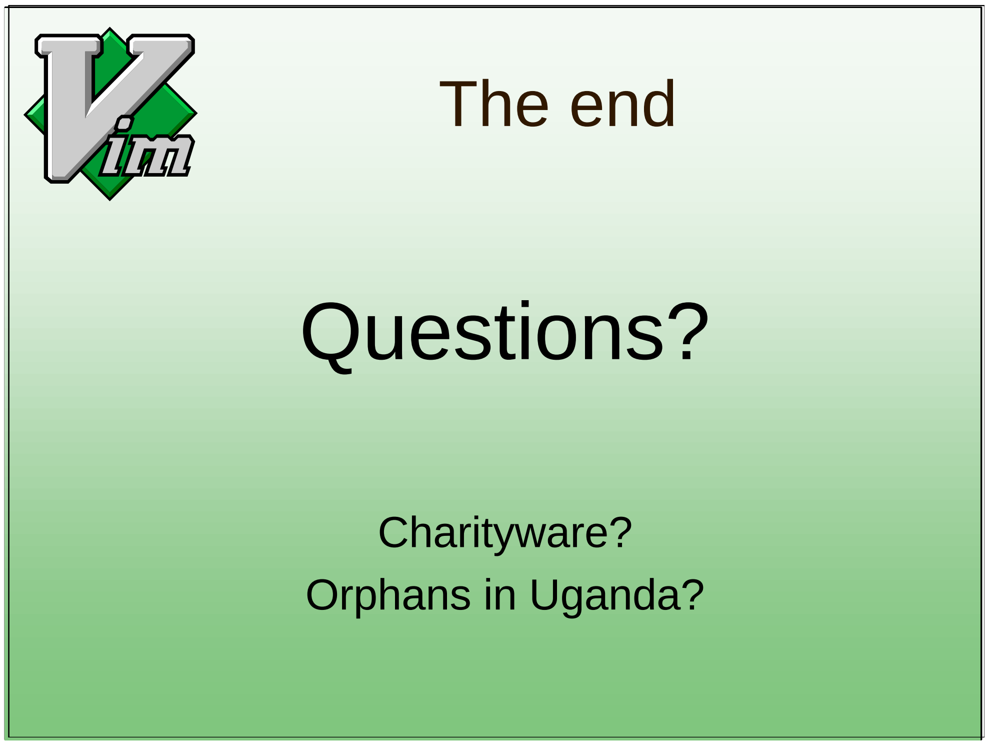
How **not** to edit effectively

You have to get the text ready right now. No time to read documentation or learn a new command.
>> You will keep on using primitive commands

You want to learn every feature the editor offers and use the most efficient command all the time.
>> You will waste a lot of time learning things you will never use.

The first remark is obvious: If you don't learn you will remain primitive.

I must warn for a pitfall when overdoing it. Learning every feature of Vim might make you the great wizard of Vim, but it will not be very effective.



The end

Questions?

Charityware?

Orphans in Uganda?

I will be around for more information.

I have some extra info about Charityware and helping orphans in Uganda.

Really the end