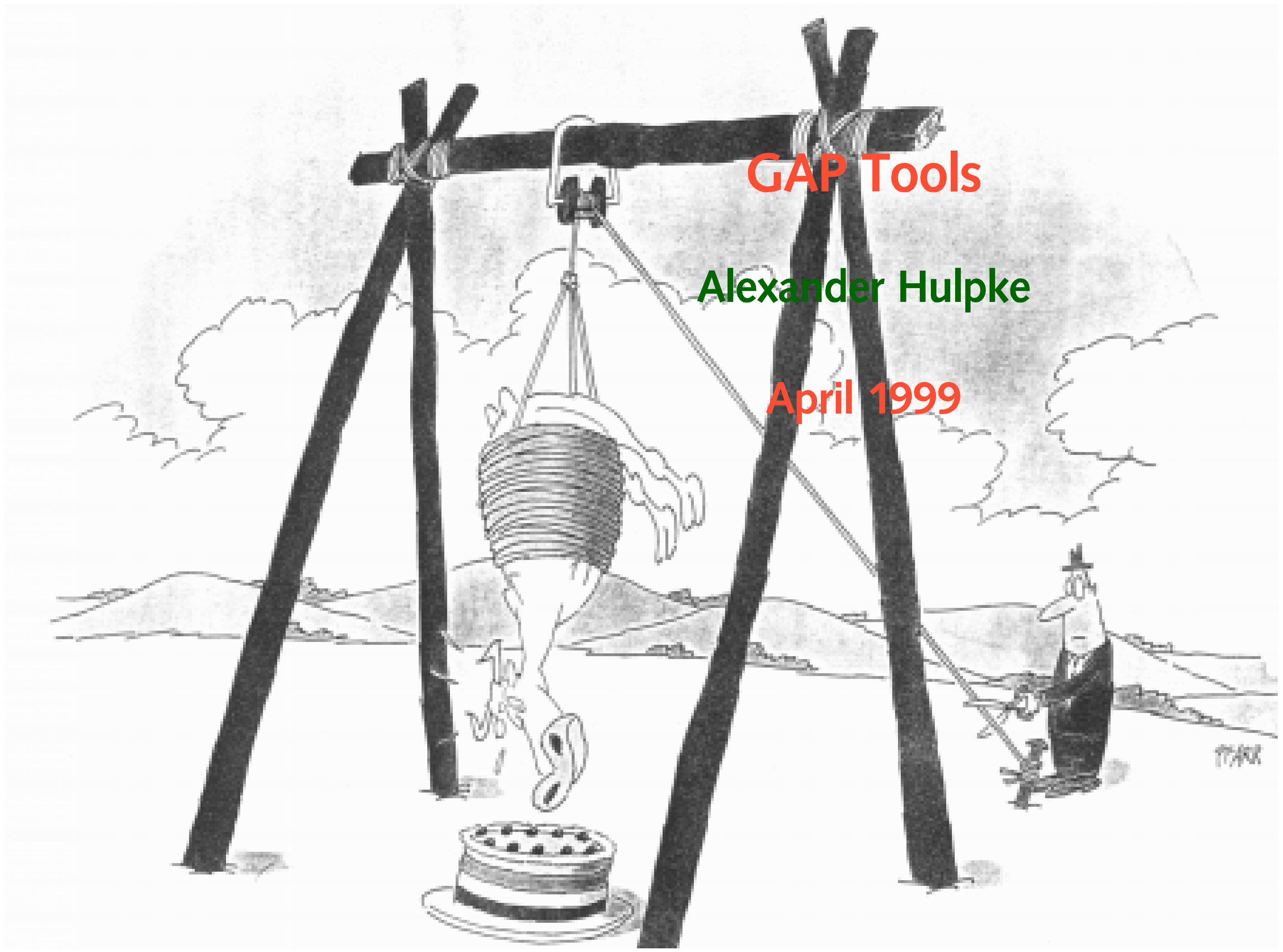
GAP Tools

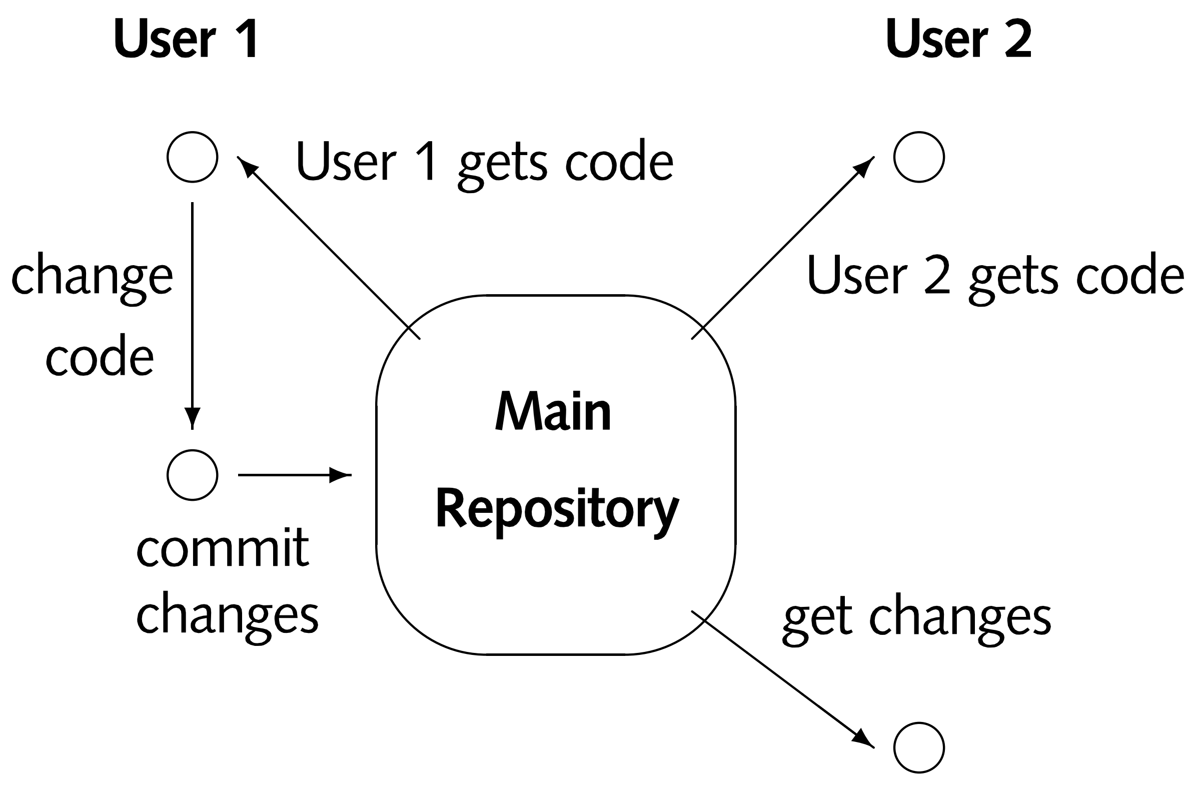
Alexander Hulpke

April 1999



CVS

CVS (`http://www.cyclic.com`) is a system for distributed code development.



All GAP code is in a CVS repository in St Andrews. We offer read access to the current development version. (This will still require a password – Ask.)

Changes only go in the repository after they are committed. If *other* changes to the *same* piece of code have been done already by another user, CVS enforces a check for compatibility by the user.

It is possible to create different development branches (we have the “current” branch and `GAP4B5`.)

CVS remembers all versions of the code, it can resurrect older versions, find differences between versions or merge changes from one branch into another.

Every committed change gets an identification text.

(Make sure you have a current CVS version (1.10 and later is safe), older ones might be prone to the Y2K problem.)

Homebuilt Tools

All other tools are somehow “home-grown”.

We use them in the development, they may be of use to other people.

Needless to say they come without any support.

All path names are relative to the GAP root directory under CVS.

Debugging Tracer

Sometimes it the `Where` display does not help to localize the position of an error (or GAP simply crashes). A desperate way to find such errors is to print “life-line” line numbers.

The shell script `etc/addebugs` adds Print statements after every “;” that count up, `etc/removedebugs` removes them again.

The scripts keep safety copies of the original files.

Manual Tester

The manual test scripts (`doc/test` take a manual chapter, extract all sections between `\beginexample–\endexample` pairs and runs the GAP input, comparing its output to the output given in the examples. It lists all discrepancies.

Tests are performed up to blanks. All output of one command is concatenated into one line.

The manual tester cannot cope with errors that will cause GAP to enter the `brk>`-loop. Such examples must be enclosed by `\begintt–\endtt`.

```
you@unix> testexample ../ref/grpperm.tex
```

Lots of not redirected screen output

```
Differences in output:
=====================
Command: h:=Group(GeneratorsOfGroup(op ));;StabChain(h);;time;
Example: 970
Output : 950
```

Manual Builder

The documentation of most functions and operations is at the place where they are created in the library:

```
###########################################
##
#O  Piffle(<woffle>,<biffle>)
##
##  computes the maximal piffle which is
##  compatible with <woffle> and <biffle>.
DeclareOperation("Piffle",[IsWoffle,IsBiffle])
```

We want this text to go in the manual with added examples.

This is achieved by the manual builder, a `perl` script `etc/buildman.pe` which replaces declarations in a skeleton file (`.msk`) by the appropriate description from the library.

The skeleton:

```
\Declaration{Piffle}
\beginexample
gap> Piffle(a,b)
<a maximal piffle>
\endexample
```

transforms the declaration:

```
#O Piffle(<woffle>,<biffle>)
## computes the maximal piffle which is
## compatible with <woffle> and <biffle>.
DeclareOperation("Piffle",[IsWoffle,IsBiffle])
```

into this manual section:

▶`Piffle(` *woffle,biffle* `)` O

computes the maximal piffle which is compatible with *woffle* and *biffle*.

```
gap> Piffle(a,b)
<a maximal piffle>
```

The manual builder is started with a configuration file that indicates which files are to be read:

```
@msfiles = ("piffle","pifflemore");
@gdfiles = ("group","kernel.g","piffle");
DIR = "../ref";
LIB = "../../lib";
```

It is also possible to define local variables which will be automatically replaced when occurring in the skeleton in double $\{\{\cdots\}\}$ brackets. (So we don't have old version numbers all over the manual.)

The skeleton files and configuration files for the GAP reference manual are in the `doc/build` directory.

There is a rudimentary `Makefile` to build everything.

HTML Converter

The HTML converter `etc/convert.pl` converts a TeX version of a manual into HTML files. The command line options -c and -s determine whether files are chapter-wise or section-wise.

When run for a share package the share package name must be given as argument to the `-n` option, otherwise the converter assumes it is run for one of the books of the main manual and will not get crossreferences right.

```
convert.pl -c -n mypkg texpath htmpath
```

The converter does not (yet) understand all subtleties of the TeX style.

It does not interpret any user-defined macros.

The converter expects a line with at least 16 percentage signs before any new `\Section` line.

Building the documentation from scratch

Part of the `ref` manual is built from skeleton files in `doc/build` and the library. Then all four manuals have to be translated with TEX, bibliography and index are created with BiBTEX and `makeindex`.

The `doc/make_doc` script does everything for you and builds local HTML documentation. (However it does not take care of changes in the library since the last documentation build. For this go in the `doc/build` directory and `touch` all `*.msk` files before running `make_doc`.)

Distribution scripts

The subdirectory `mkdistrib` contains scripts that build a new distribution from scratch.

Most of this is only of internal interest, but it might help to get started with a CVS version on your own.

Zoo Unzoo

We use the `zoo` compressor because:

- Historical Reasons – Changing would involve lots of work.
- We provide a free (GPL) source and binary for all platforms. (The versions for Win and Mac of other compressors usually are shareware and require registration.)
- We use a slight extension of the `zoo` format to distinguish binary and text files (CRLF problem).

Every file in a GAP `.zoo` archive has a comment `!TEXT!` or `!BINARY!` that tells whether a CRLF translation should take place. The `unzoo` decompressor we provide uses these comments.

If you submit a share package in another format we must repack it with `zoo`. You save us much work if you wrap your package directly this way.

Bugzilla

Bugzilla is a bug tracking system developed by the `Mozilla` folks. We use a modified version to keep track of bugs and as a “wish list”.

These lists are for our internal use, we do not guarantee that entries will be considered.

To access these lists you need a web browser (with cookies turned on!), the URLs are:

```
http://www-history.mcs.st-and.ac.uk/gapbugs/
http://www-history.mcs.st-and.ac.uk/gapwish/
```

The first use forces you to register for a password.