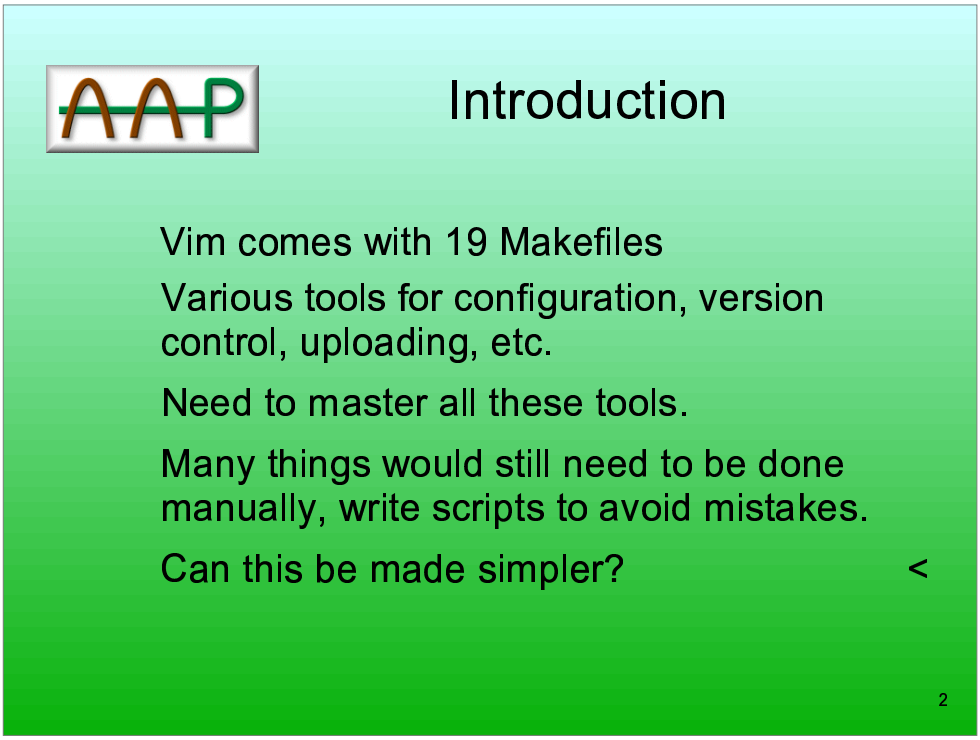
A-A-P: A Software Build Facility For The Internet Era

Bram Moolenaar
OSCON 2003

Presentation given at the O'Reilly Open Source Convention, July 7 – 11 2003, Portland, OR, USA.

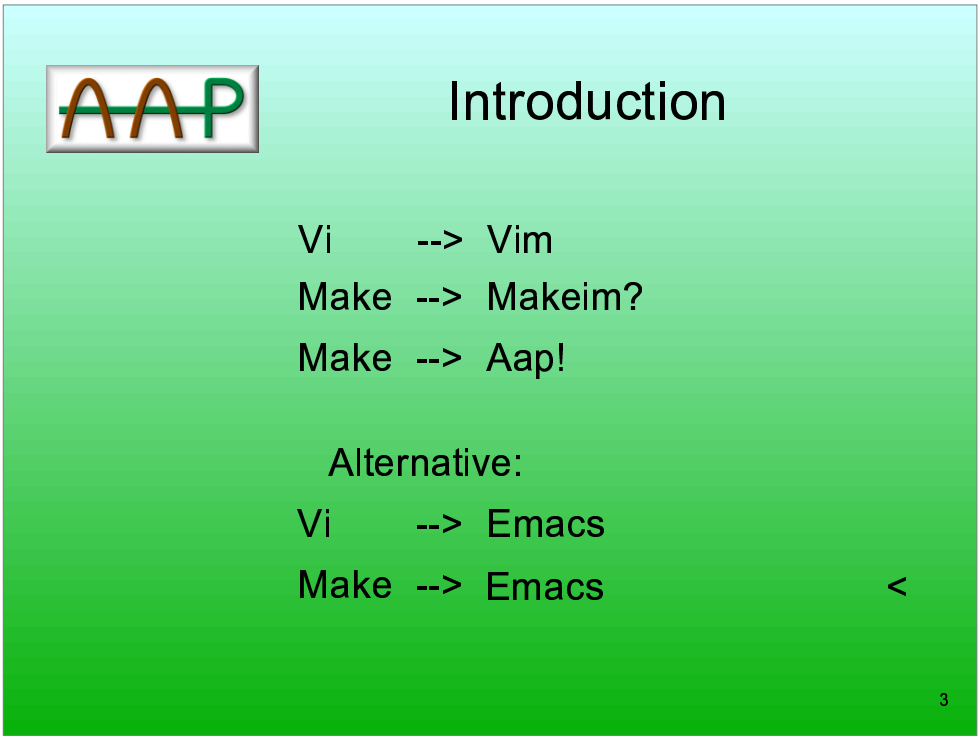
Author: Bram Moolenaar <Bram@A-A-P.org>



You probably know me as the creator of the text editor Vim – Vi IMproved. I have attempted to make it run on many different systems. For this a lot of work had to be done. On Unix autoconf can be used to adjust to various properties of the system. Unfortunately, this is not available on other systems. This resulted in a large number of `Makefiles and #ifdefs in the code.

To automate all the complicated steps of building and producing a distribution many different tools are needed, each with their own arguments. Currently this only works on Unix, because it is too much work to also support other systems.

The result is a combination of Makefiles and shell scripts that only I fully understand.

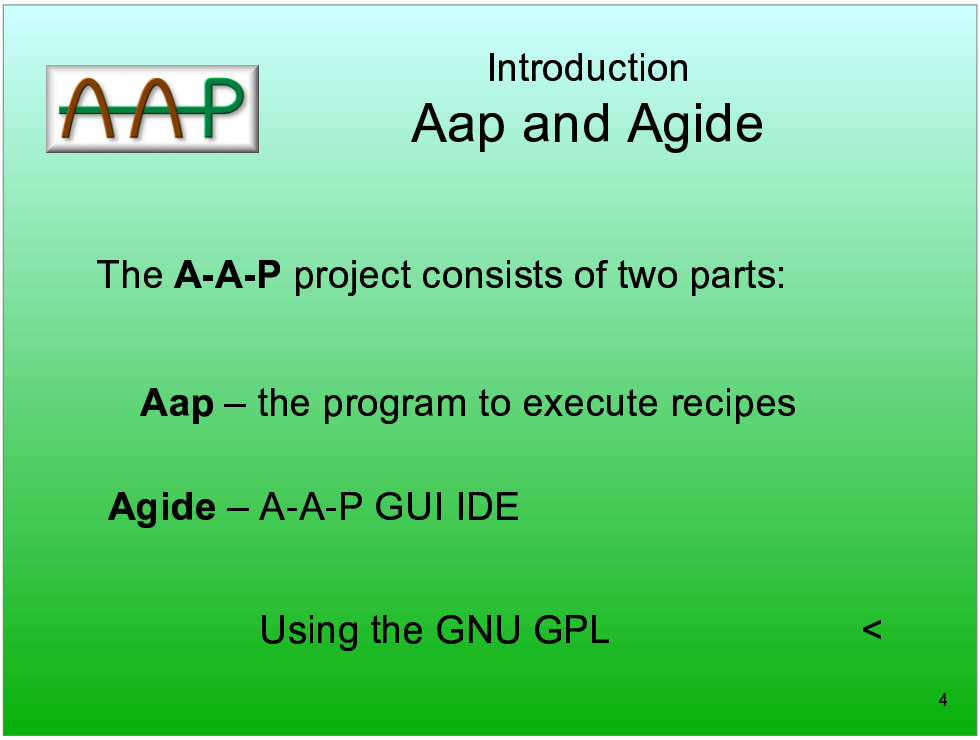


Out of Vi came Vi IMproved. So what do you get when you improve make?

“Makeim” doesn't sound very nice and is too long to type, so I choose to use “Aap”.

If the improved way of editing for you means making the step from Vi to Emacs , then what would you call an improved make? Well, Emacs as well, of course. You might argue that Emacs is not a very good build system, but that applies to text editing as well :-).

Anyway, I decided to start the A-A-P project to create a nice build facility.



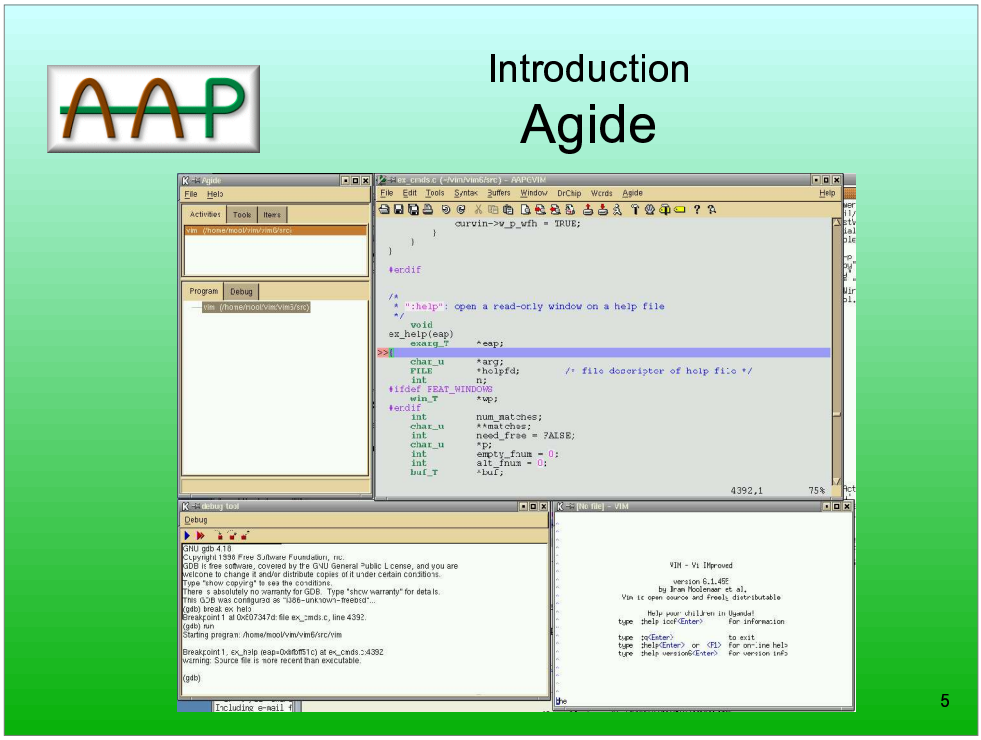
The name A-A-P does not stand for anything yet. It does make the project appear as the first entry in sorted lists!

Aap is a program, written in Python. The script file it executes is called a recipe. What a recipe looks like will be explained further on.

Agide is a framework. The core is a Python program with WxPython. Many different programs are bound together, for example Vim and gdb. The Agide project file is an Aap recipe.

All software of the A-A-P project is using the GNU GPL.

This presentation is about Aap. But I will first give you some idea what Agide is.



This slide shows Agide being used for debugging Vim. This is what Agide is capable of right now. You can set breakpoints, single-step through the code, etc. I actually use this for my work on Vim now.

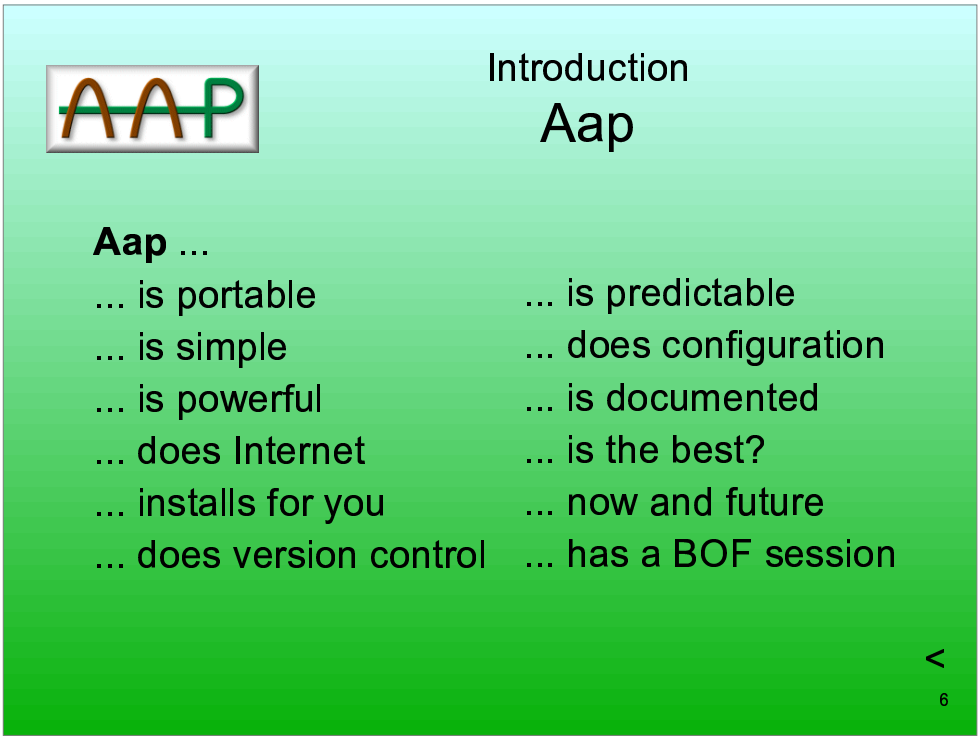
The left-upper window is the Agide navigation window. It can be used to browse the project, running tools and select an opened window.

The right-upper window is Vim, being used to edit a source file. You can set breakpoints here and see the value of a variable by pointing at it (balloon evaluation).

You can type any gdb command in the gdb window (left-lower window).

A separate xterm is used to run the program in (lower right window), so that its output doesn't clobber the debugger output.

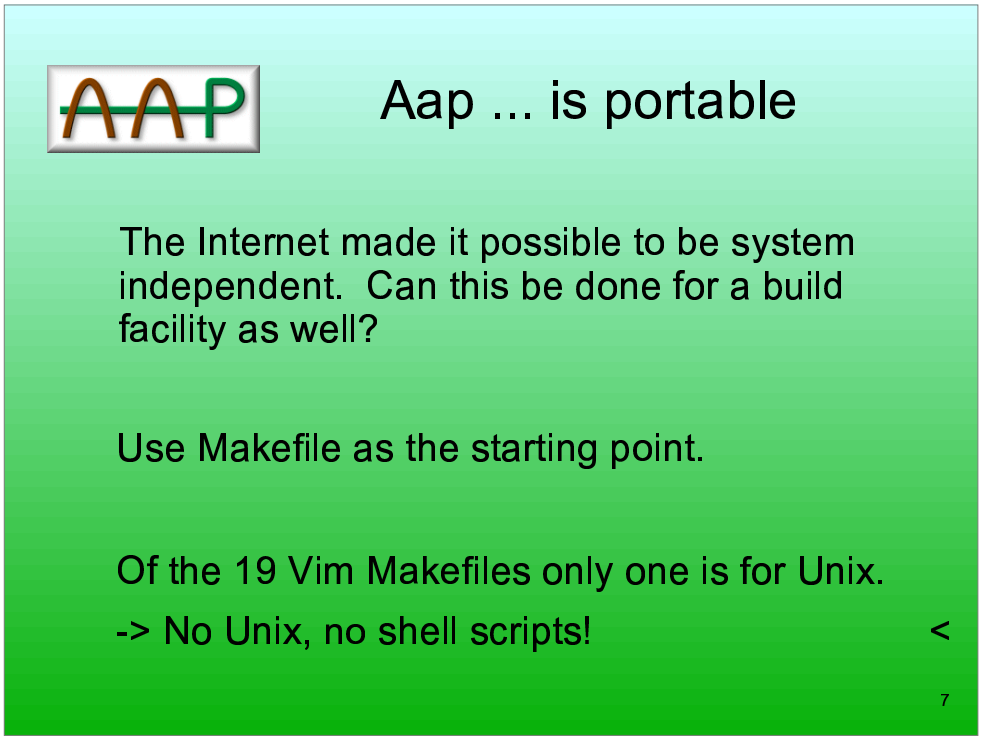
It should be possible to use another editor without changing the debugger tool, and to change the debugger without modifying the editor. The tools do not interface directly, Agide takes care of connecting the various tools together.



Aap has many features. I have attempted to mention the most important ones, but the list is already very long. I will have to skip a few parts to fit the time available.

The reason so many features are included is that it is not sufficient to provide one more feature than another build tool. Aap must provide a solution for all issues that developers encounter. When an essential feature would be missing, you cannot use Aap.

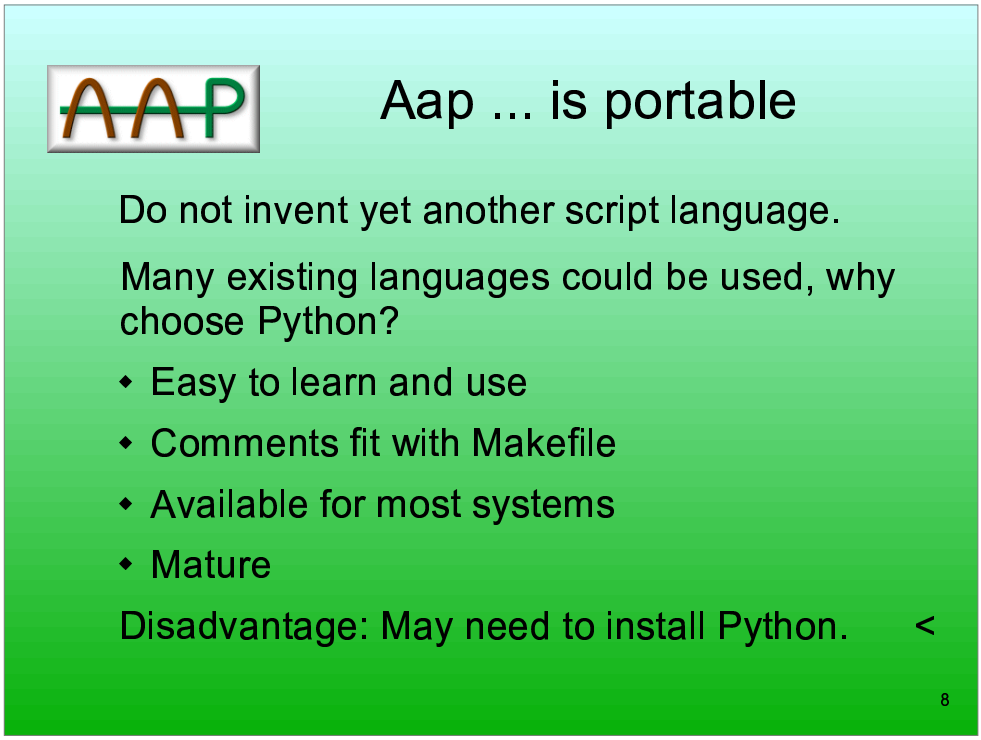
In other words: Everything you always wanted in a build facility, but were afraid to ask for.



Since the Makefile is a very popular file format, it has been used as a starting point.

Many different Makefile dialects exist and trying to be backwards compatible conflicts with the goal of adding new features in an “easy to use way”. Therefore no attempt was made to be backwards compatible.

To be portable to non-Unix systems, use of shell scripts must be avoided.

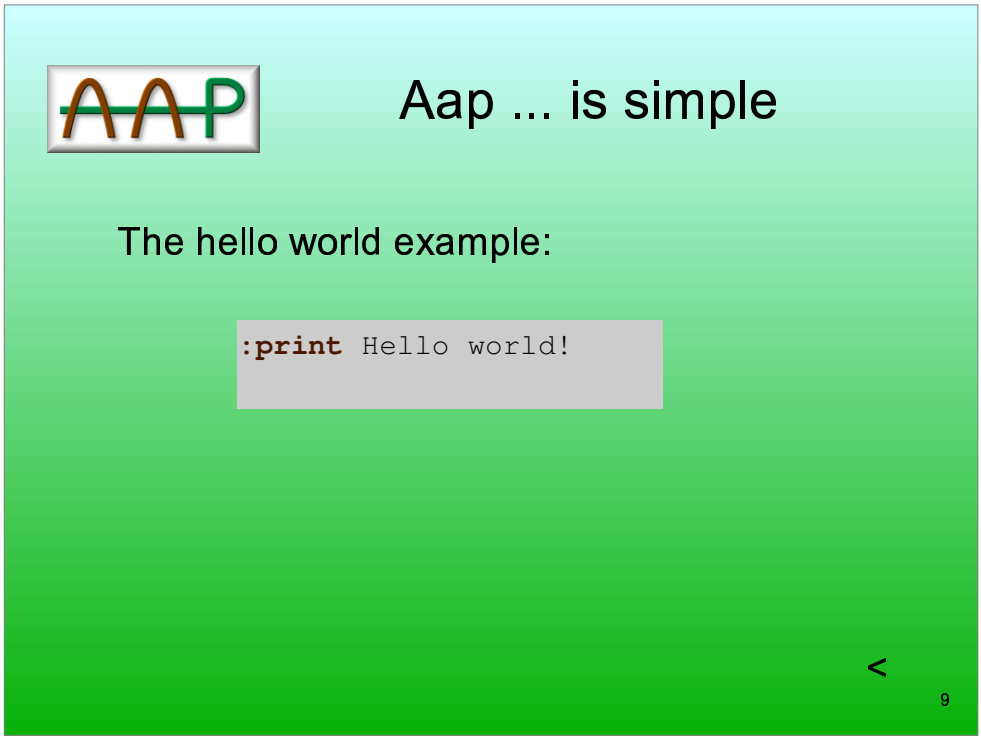


One might argue that language xyz is better than Python. Unfortunatly it's a different language for everyone. I had to make a choice. As it turned out, Python works quite well.

Just like Python was made to replace shell scripts and C programs, Aap was made to replace shell scripts and Makefiles.

Python may need to be installed on systems where it is not available. Other script languages have the same problem, there is no script language that is supported on both MS-Windows and Unix.

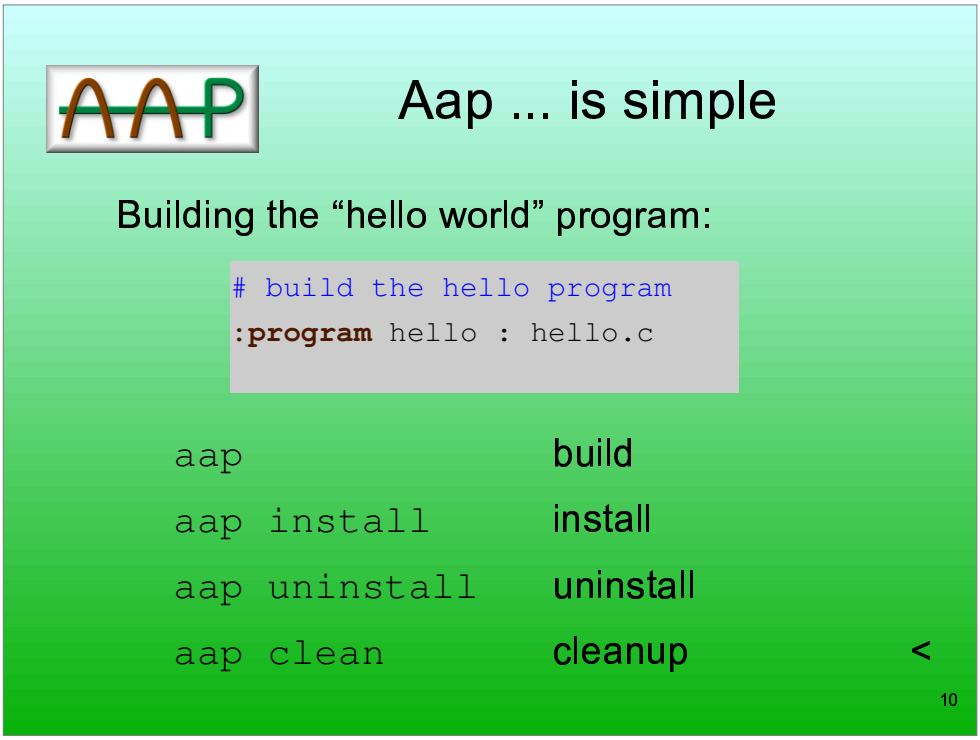
OK, so now we have a file format that is based on a Makefile and uses Python instead of shell scripts. This is called a recipe. Next we will see how this is used.



This is really a silly example.

What this shows is that recipe commands start with a colon. That makes them easy to recognize

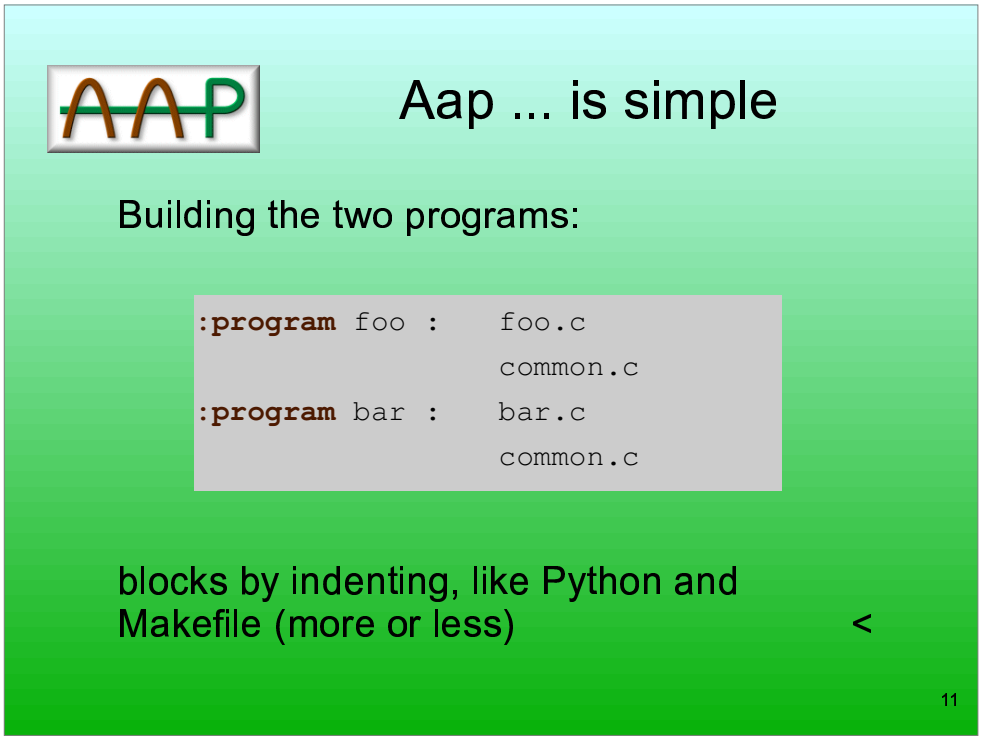
Note that there are no quotes around the text.



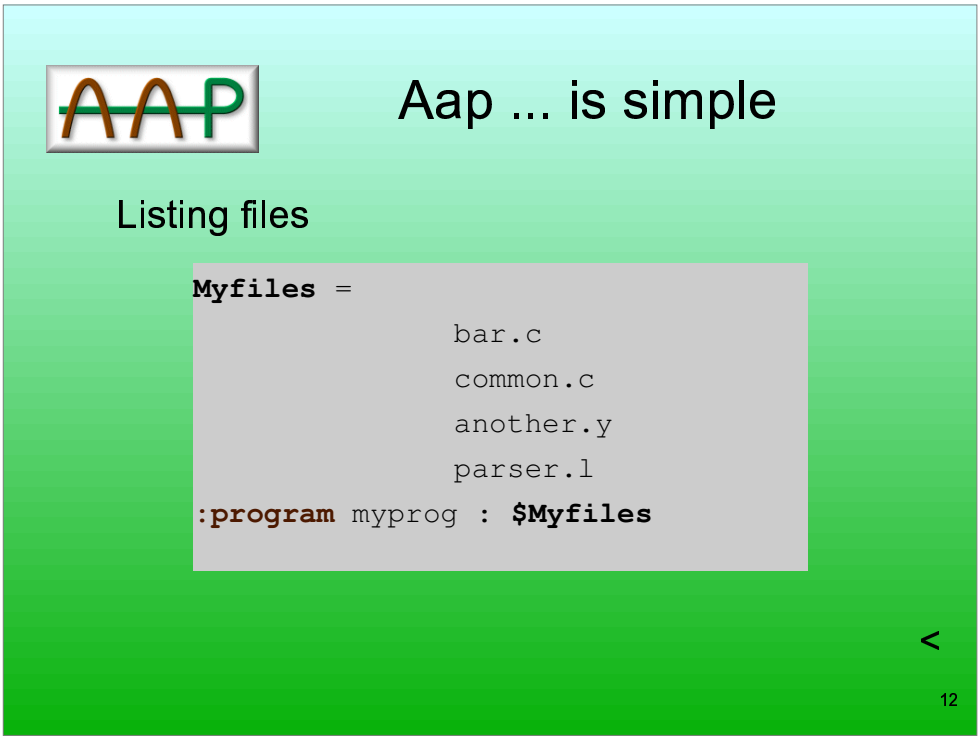
The “:program” command specifies the name of the program to be build and the source files that it is to be made of.

Write this recipe as “main.aap” and execute the “aap” program without arguments. Aap will figure out dependencies, compile hello.c into an object file and link it into the “hello” program.

Furthermore, Aap will offer the functionality to install, uninstall and cleanup. This works more or less like automake. The $DESTDIR and $PREFIX variables are used to specify where the files are installed. Currently “aap install” only works for Unix.

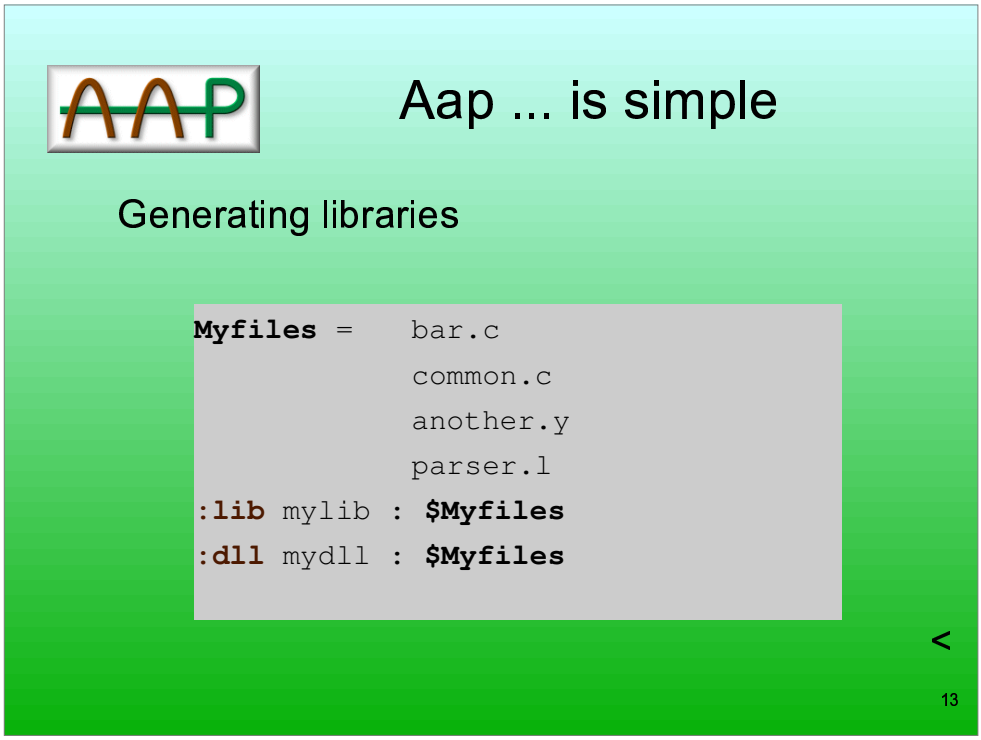


Note that no quotes or line continuation is used. This makes it easy to list files and hard to make mistakes with a backslash somewhere.



The list of files is easy to sort, no need to fix the backslash on the last item.

Note that yacc and lex files can be used as well. They are processed to generate a C file automatically.



Libraries can be created in the same way as a program.

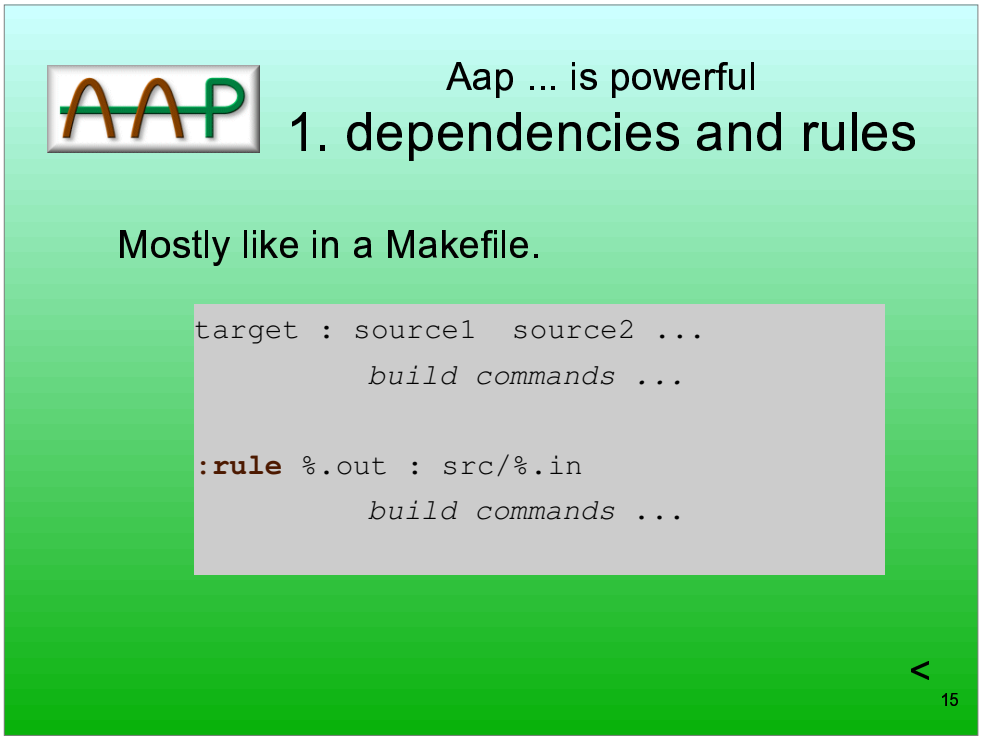
Aap will use the default way to create a (shared) library with your compiler. That is, if it knows how to do that. The idea is that the knowledge is added to Aap, so that not every user has to figure it out himself. And it makes recipes portable.

In the future more commands like “:program”, “:dll” and “:lib” will be added. Perhaps a “:pdf” command can be used to generate a PDF file from various file formats.

Real-world recipes will be more complicated, since you mostly want to do more than the obvious things. For that you need the basic mechanisms that the recipe offers.

These are the most important low-level mechanisms that can be used.

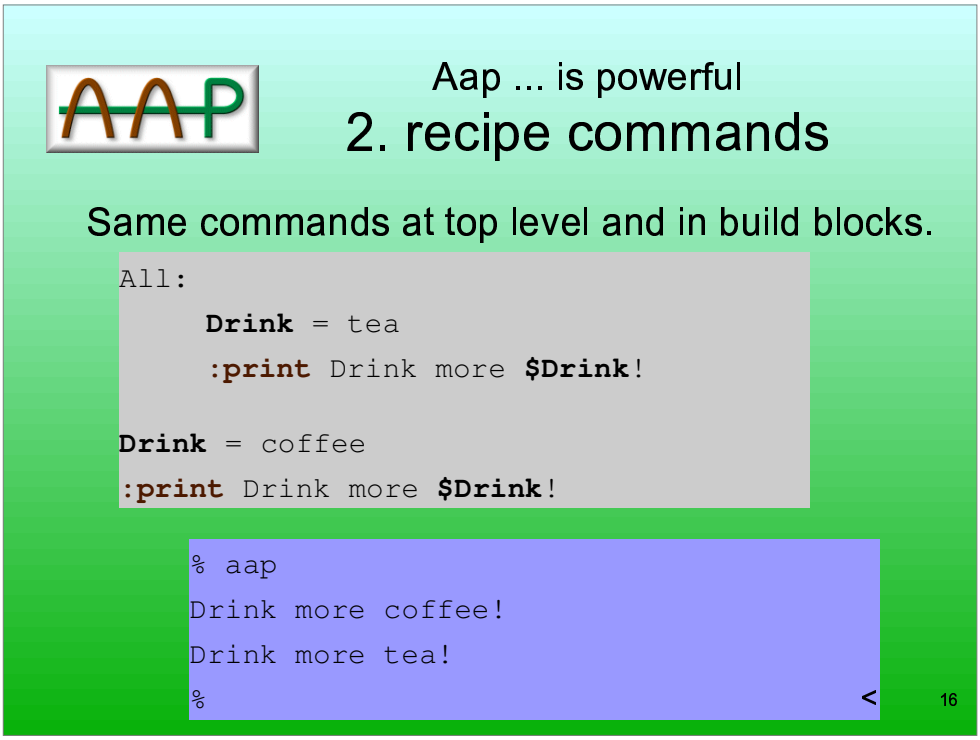
I will explain each of them briefly.



The dependencies can be mixed with the high-level commands like ":program". That can be used to do part of building in a different way.

This is typical for Aap: offer a high-level solution that is easy to use for most purposes, offer a low-level solution for specific purposes.

--- Keep simple things simple, make complicated things possible ---



The same commands can be used at the top level of the recipe and in build blocks. This is quite different from a Makefile, where at the top level only assignments are allowed and in build blocks only shell commands are used.

There are a few exceptions. For example, dependencies and rules can only be defined at the top level.

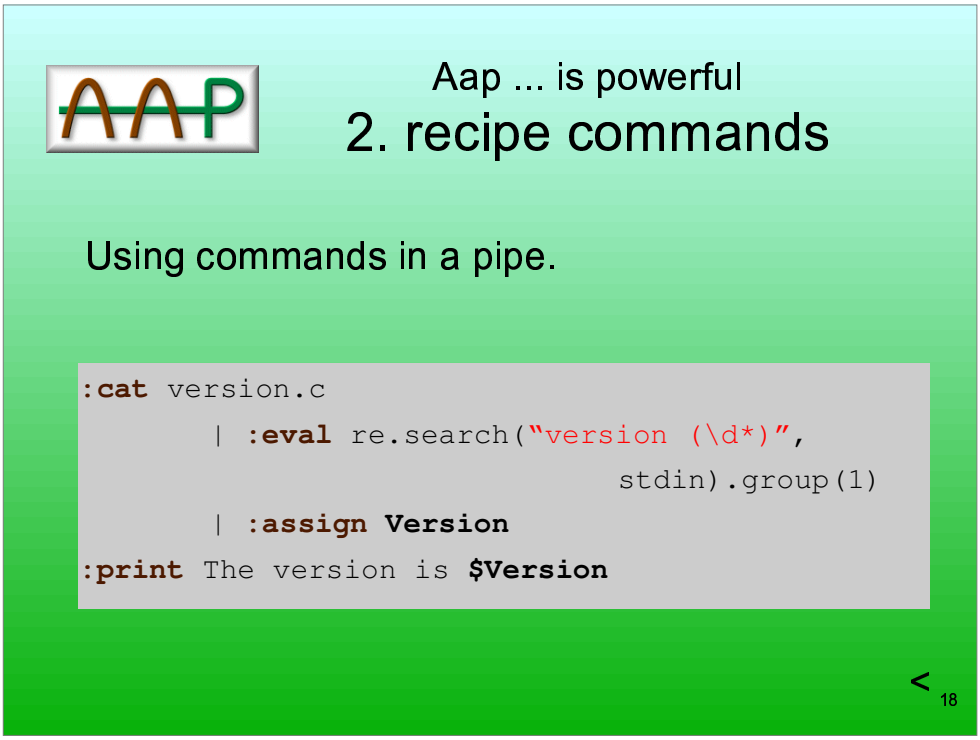
Just like in a Makefile, first the whole recipe is read and commands at the top level are executed. Then the dependency graph is followed, depth-first.

Variables actually may exist in different scopes, but that is out of the scope of this presentation.

Of course it would be possible to write these commands in Python, but this quickly results in complicated code to handle errors and handle less obvious desires.

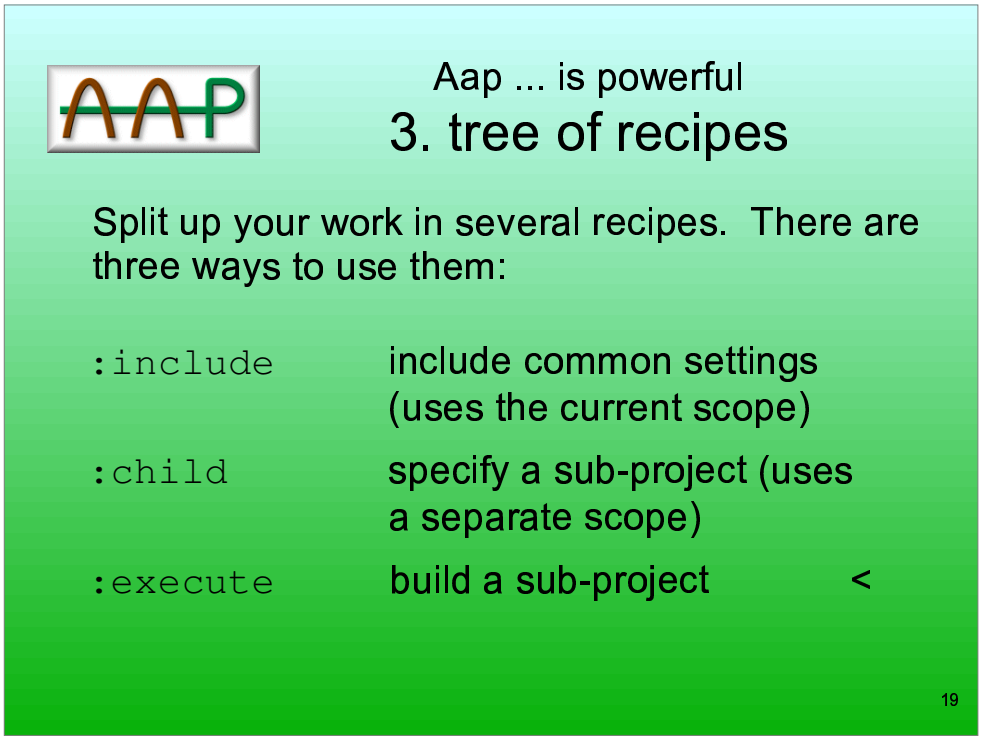
The copy command is especially powerful, since it accepts `URLs. This makes it possible to copy a file from one machine to another (provided you have access to both machines!).

The “:sys” command is needed at the lowest level, for example to start the compiler. Mostly this will be taken care of by actions, this is explained further on.



This shows that the basic mechanisms of the shell are available in the recipe.

Note that this is about the most complex thing you can do with recipe commands. If you want more you have to use Python script.



For larger projects it is convenient to split up the work in sub-projects. You can make one toplevel recipe “main.aap” that refers to the sub-projects with the “:child” command.

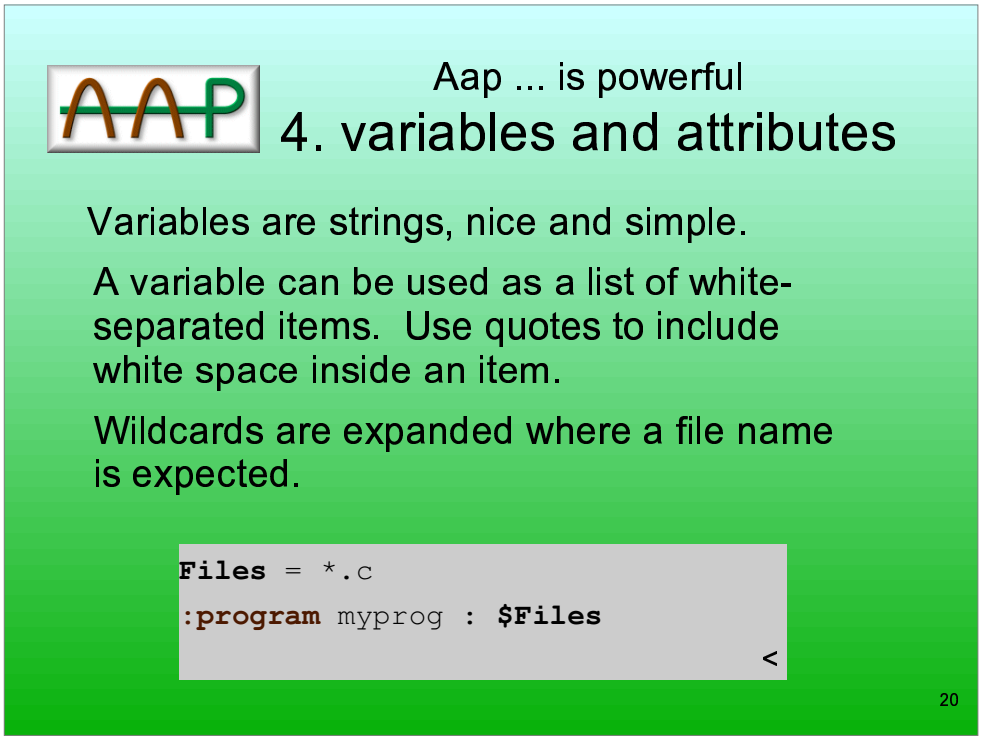
File names used in a parent or child recipe are always relative to where the recipe is located.

The “:include” command includes another recipe in the current scope. This is useful for common settings, not for sub-projects. It does not change directory.

The “:execute” command can be used to build a target of a sub-project directly. It is almost like executing Aap on a recipe.

Except for “:execute” all recipes are read and parsed before dependencies are handled. This takes care of most complicated dependencies between sub-projects.

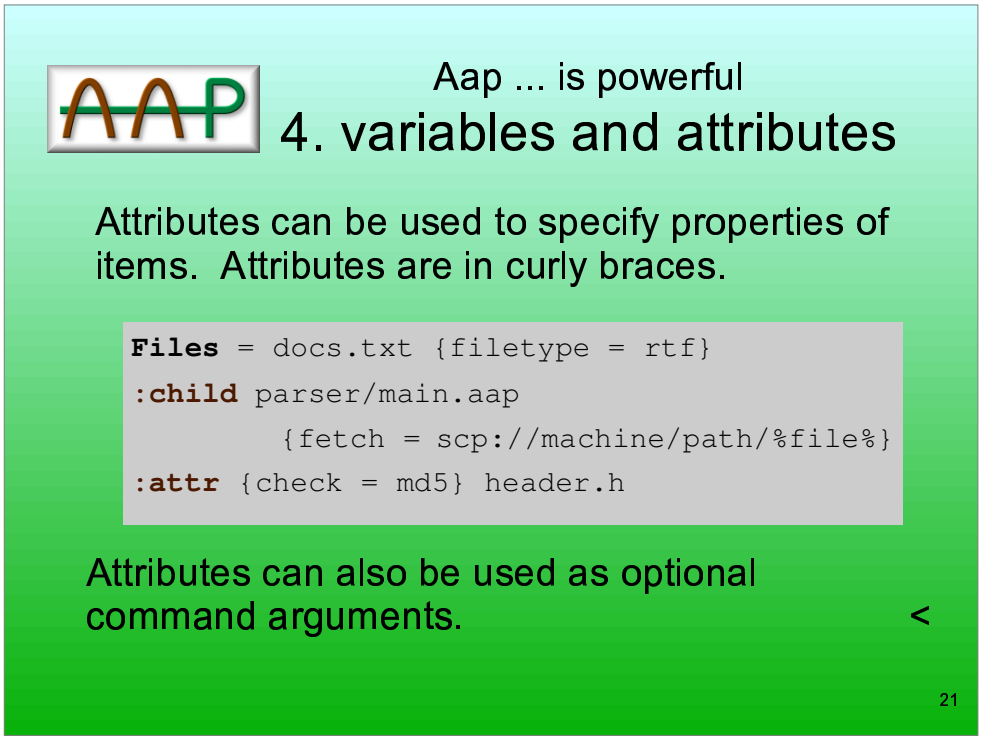
Note: recipes do not need to exist, they can be automatically downloaded the moment they are used.



Just like in a Makefile variables are strings. A difference is that $VAR items are evaluated right away, this is what most programmers expect. Postponed evaluation is also possible with the “$=” assignment.

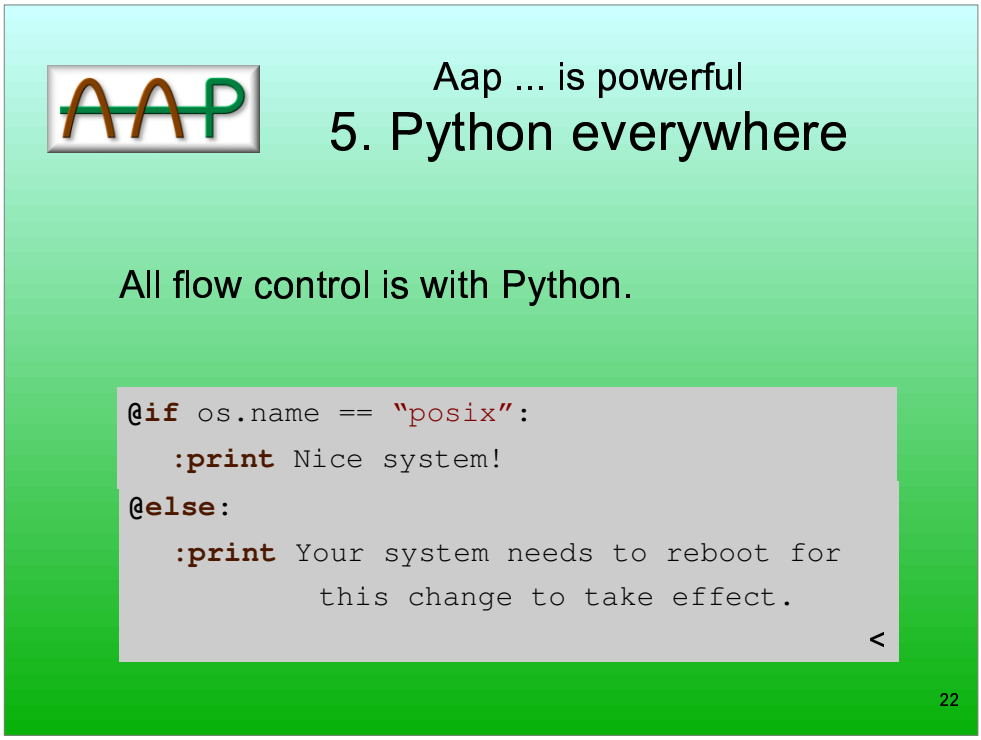
Quotes are used to include white space in an item. How to include a quote then? Use double quotes around a single quote and single quotes around a double quote. Just like Python. But no backslashes are used, to avoid trouble with MS-Windows filenames.

The special meaning of a wildcard can be avoided by putting the character inside square brackets: [*] Most other special characters can be given with the form $(x). For example, $($) stands for a dollar.



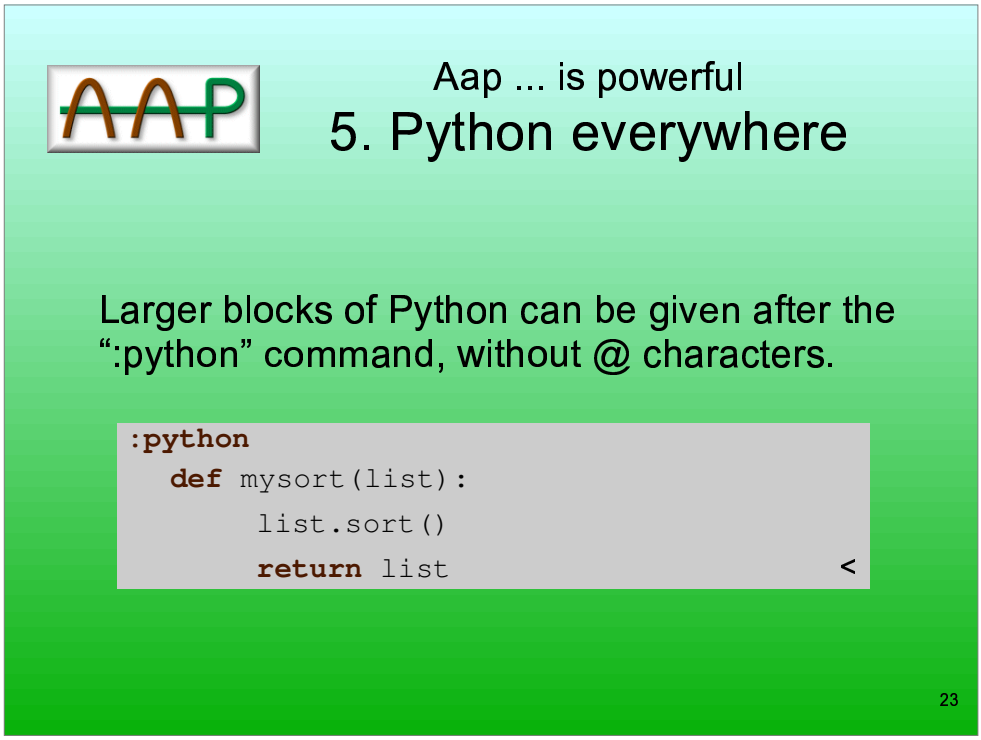
The use of attributes provides a very powerful mechanism. This makes it possible to extend the functionality of Aap in a uniform way.

Attributes can given as part of an assignment, as part of the arguments of a command and explicitly with the ":attr" command.

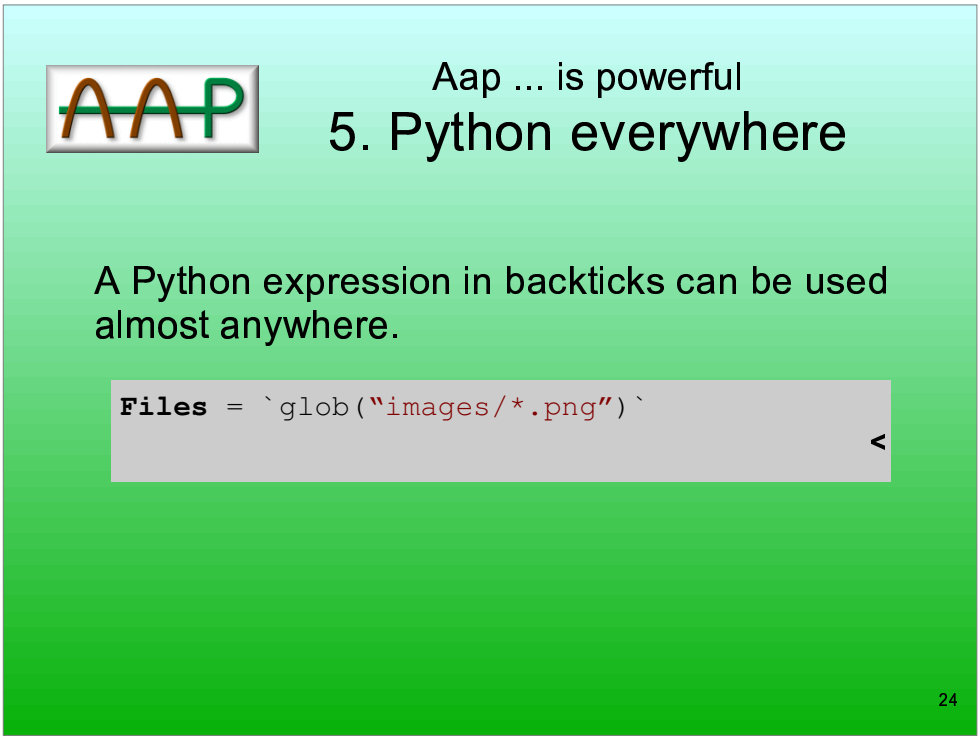


A line of Python script is preceded with the @ character.

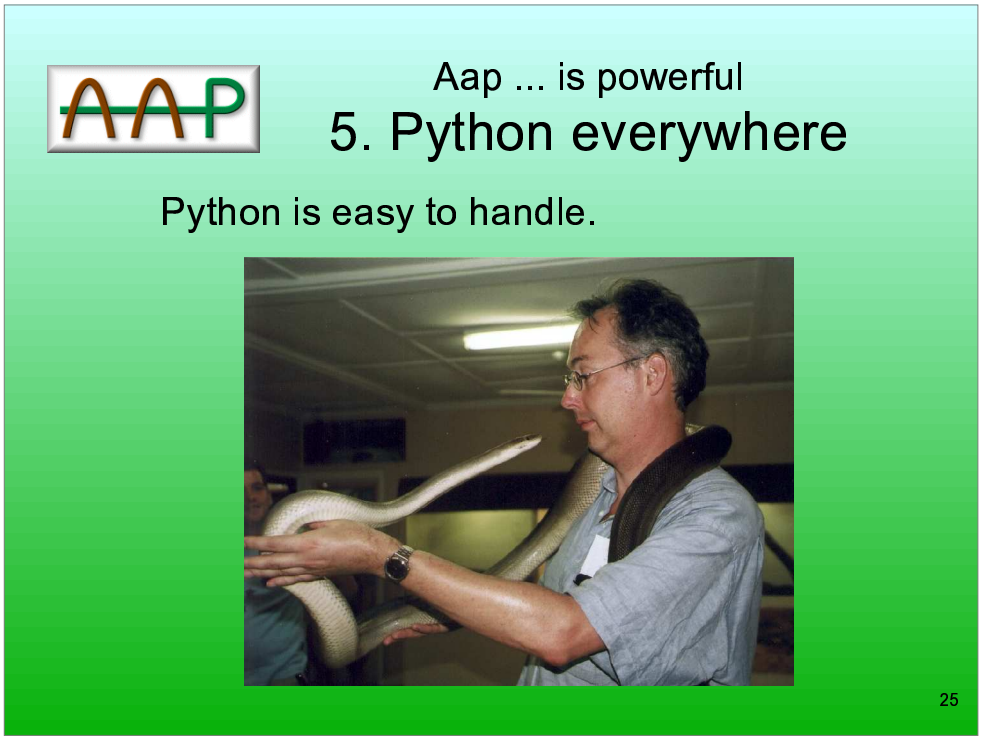
Notice that Python and recipe commands can be mixed as you like. The indent is used to define command blocks in the same way as in Python.



If your python code gets really big you can also put it in a separate file and "import" it.



The difference with using wildcards directly is that these are only expanded where a filename is expected. For an assignment any string can be given, thus wildcards are not expanded. The Python glob() function is used here to explicitly expand wildcards.

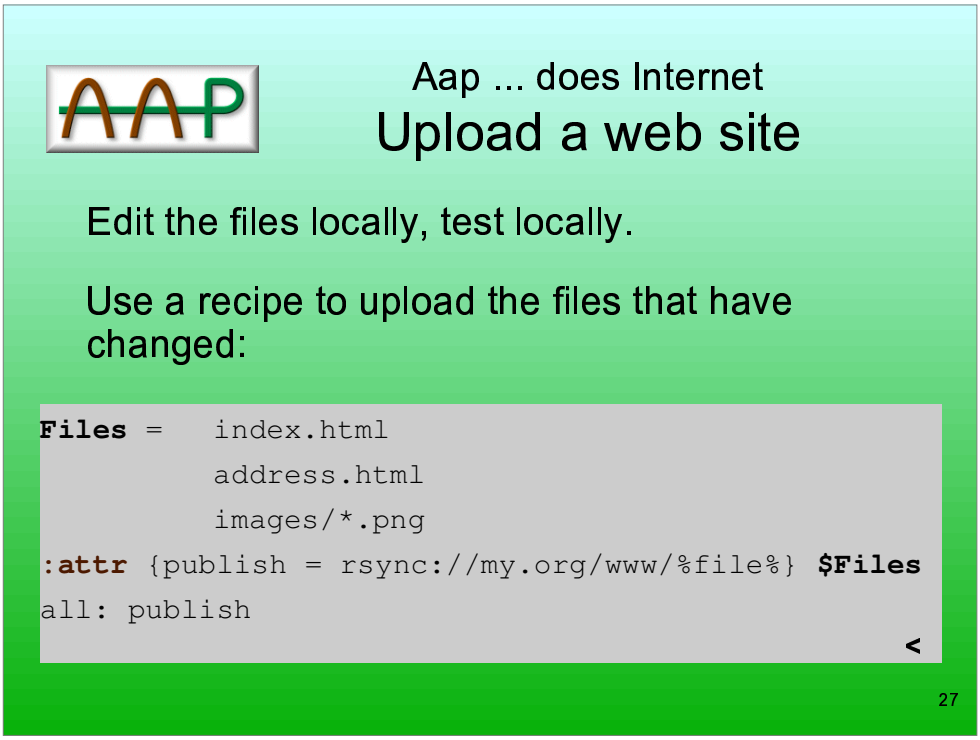


Don't try this at home!.

When executing this recipe Aap will download the specified files before building.

Making a local copy is needed for reliability. There are several settings to specify when a new copy is to be obtained. In this example the file is obtained if it does not exist or when updating is explicitly requested.

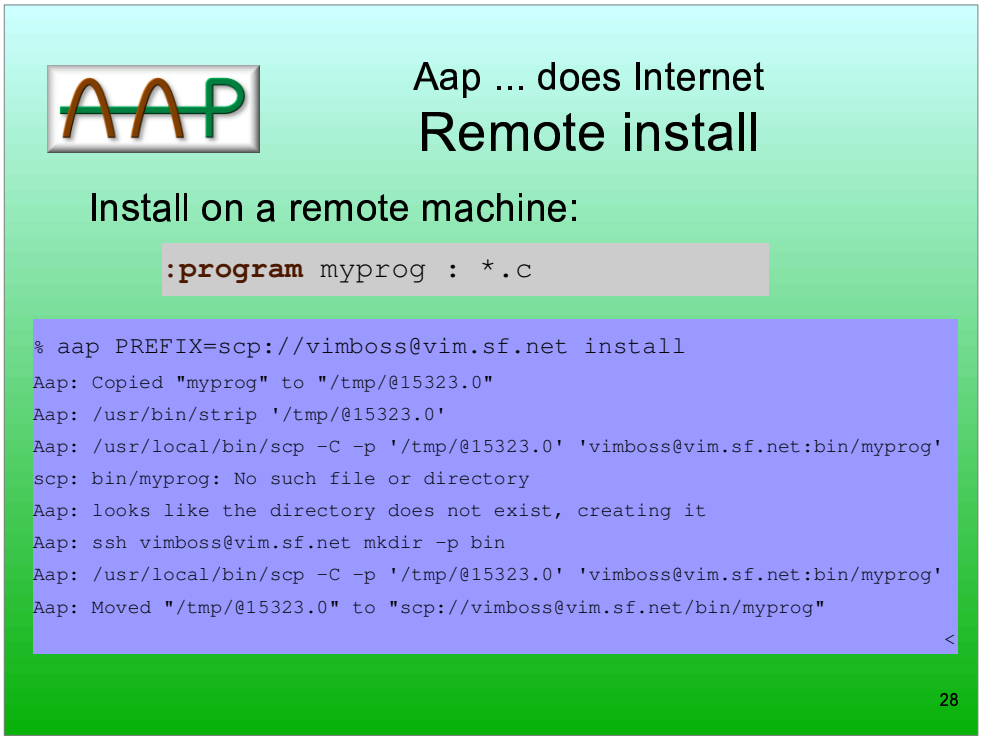
Besides http:// you can currently use ftp://, rcp://, scp:// and rsync://.



When executing this recipe Aap will figure out which files changed since the last time uploading was done. Thus you don't need to remember which files you changed.

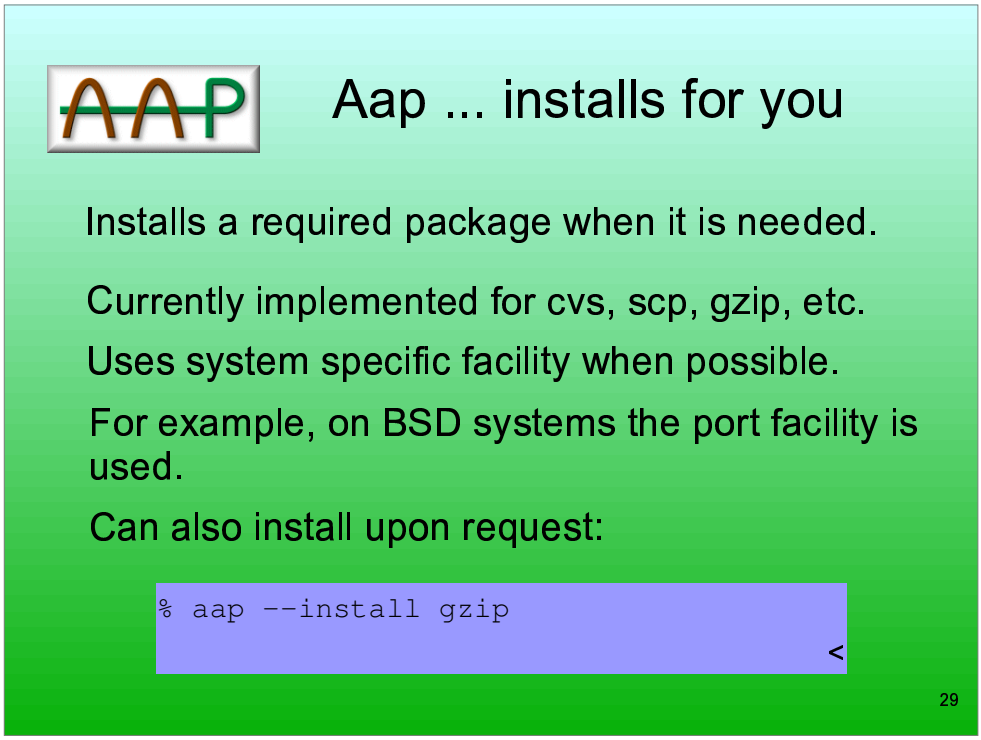
Execute this recipe with “aap publish”. Actually, running Aap withouth arguments also works, since the “all” target depends on the “publish” target.

Rsync is used here, so that small changes to big files are transferred quickly. I use recipe like this for most of my own web sites.



Since this looks too simple to be true, I have executed the recipe and you can see the actual output of installing a program remotely.

Note that the “bin” directory didn't exist yet and Aap created it automatically.

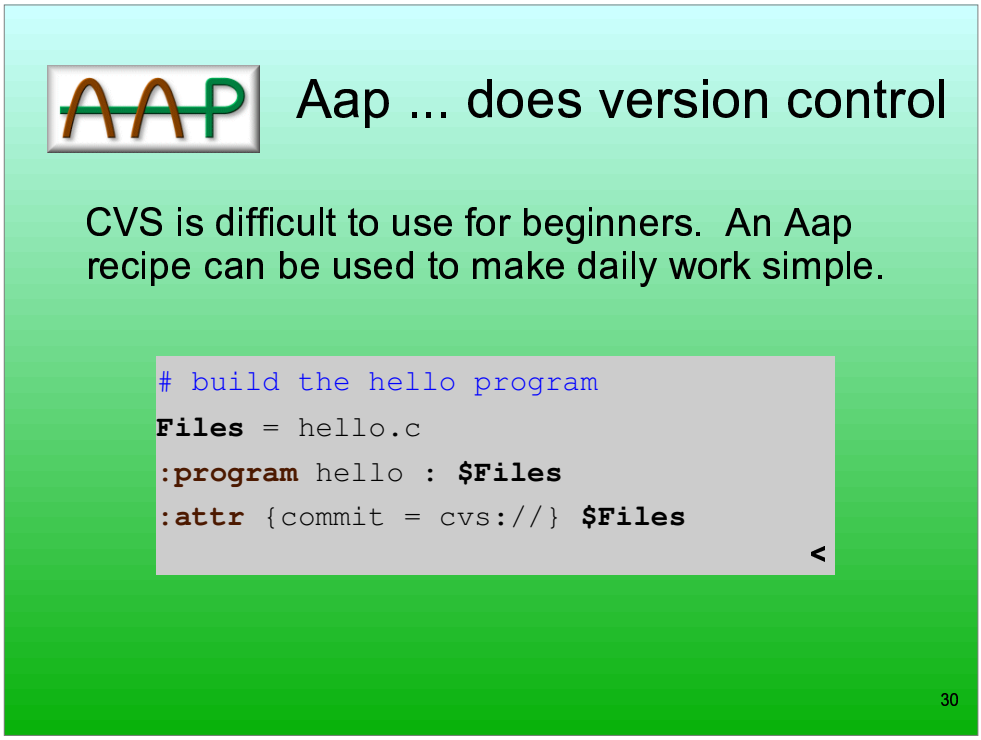


There is no version handling yet. Requesting a specific version of a package and updating to a new version of a package is still to be implemented.

How it works:

- The install facility obtains a bootstrap recipe from the Aap web site.
- Depending on the system, another recipe may be downloaded and executed.
- The recipe contains the download and installation instructions (simple or complicated).
- Possibility to redirect to another site, so that each package can be maintained by someone else.

Security is still to be taken care of. A system with checksums could be used. The issues involved are not different from other automatic update facilities.

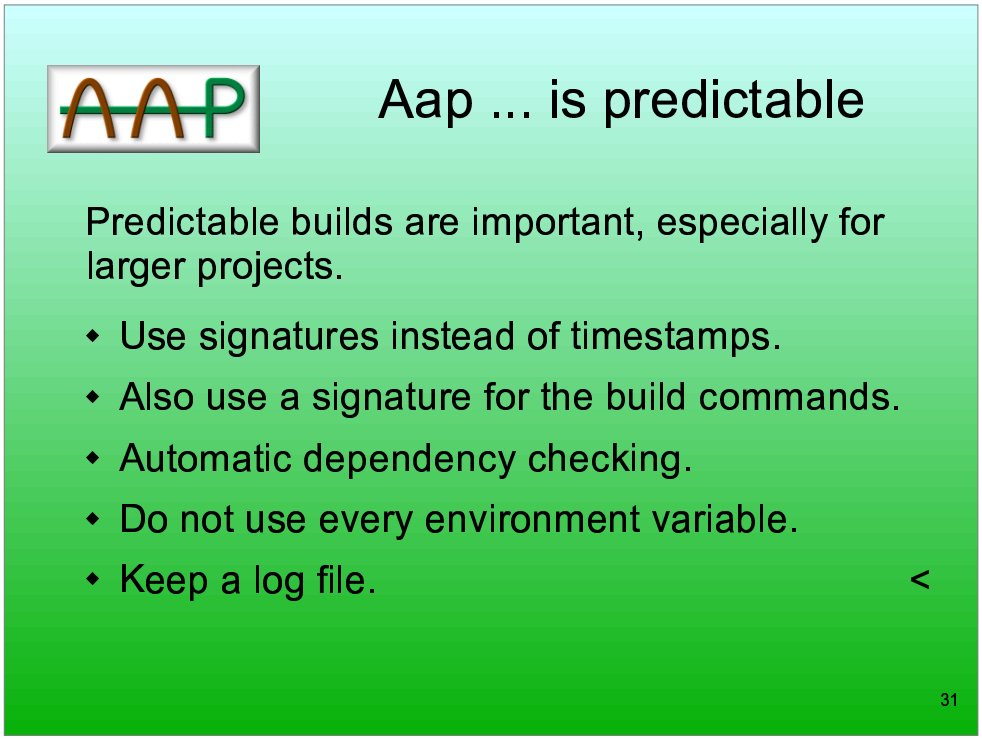


Currently starting a project in CVS must still be done directly with CVS commands. Since this requires a sequence of actions and mistakes can be made this could also be done by Aap or Agide.

Creating a repository also needs to be done directly with CVS, but that probably isn't a task for Aap. This could be added to Agide.

When a file is added and Aap is run with the “revise” argument, Aap will add the file to the CVS repository. Deleting works the same way.

It is very important that the list of files in the recipe is correct. Otherwise files could be added that don't belong in CVS. Especially watch out with wildcards.



You can also chose to use timestamps if you prefer.

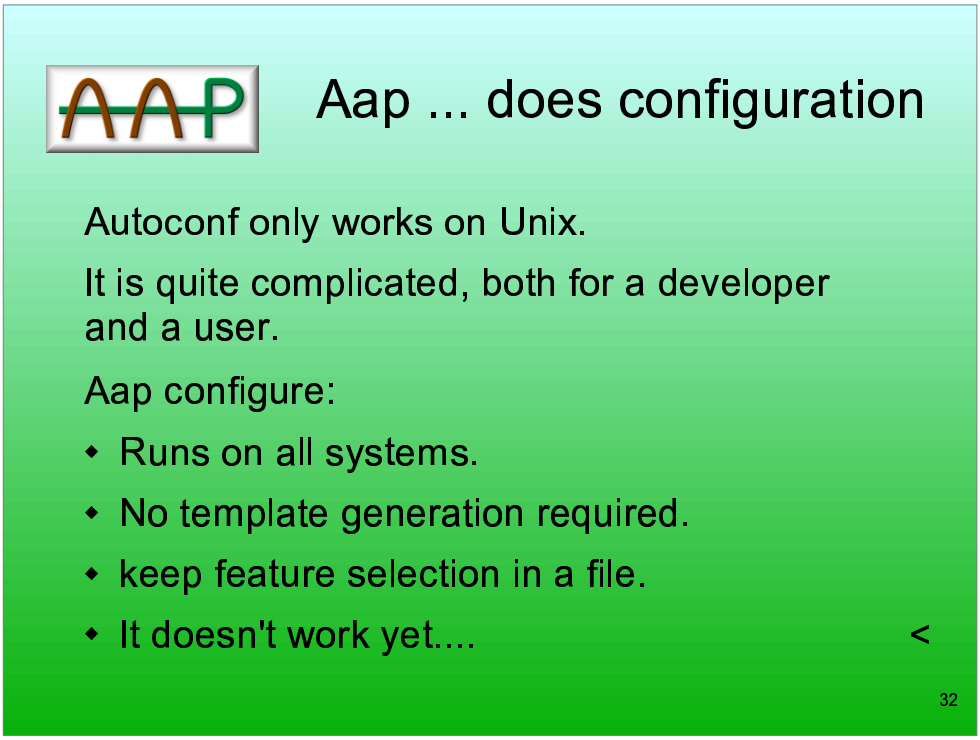
When a build command or just an option is changed, the targets involved will be rebuild. Especially useful when changing compiler flags.

Automatic dependency checking currently works for C files. Support for other languages can be added by the user.

I recently had a strange problem with running make to build a program: It insisted in linking with Python, even though I was certain I had not enabled the Python feature. After adding a few echo commands I discovered the Makefile was using $PYTHON, which happended to be set as an environment variable.

Aap does use variables like $PATH, but not $CFLAGS or $CC from the environment. Otherwise giving the recipe to someone else is not sufficient for him to reproduce what you do with the recipe.

The log file can be used to find out what went wrong, even when you forgot to redirect the output of Aap. It can also be used to review the dependencies between the files.



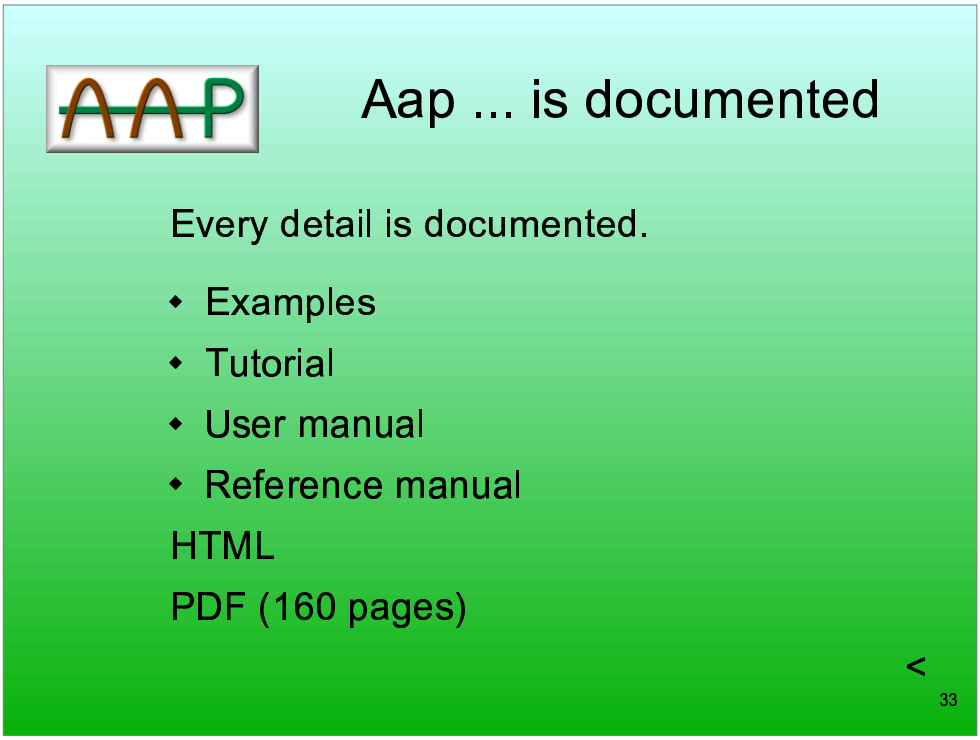
“All systems” means: all systems where Python runs.

The idea is to run tests more or less like configure does this. But not using shell scripts, which are difficult to write in a portable way.

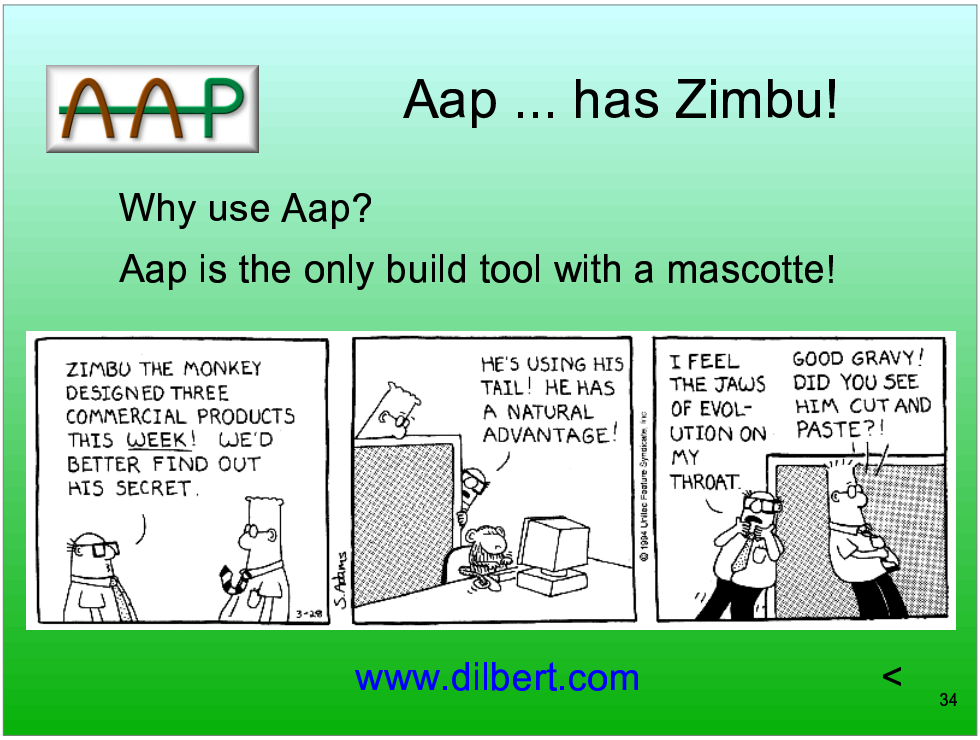
The configuration will be done with recipe commands. You can make a separate recipe to do the configuration, or mix the commands in between building. Avoids the need to do configuration when cleaning up.

Most people write a short shell script to store their configure arguments. Aap supports this in a uniform way, so that you can use a GUI for this (not implemented yet).

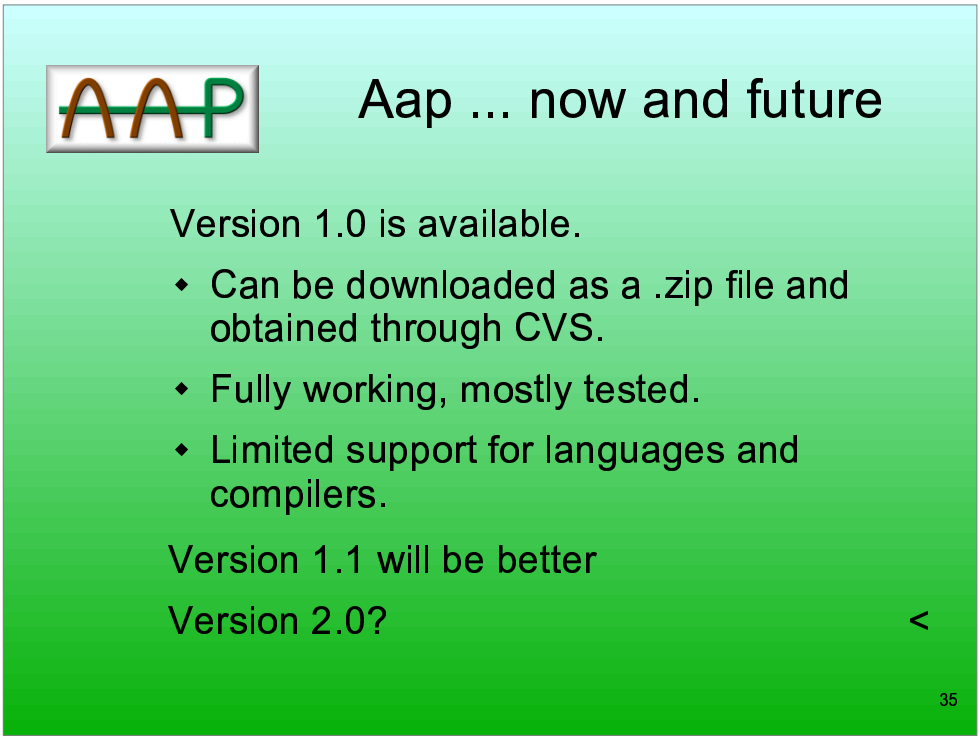
Joerg Beyer has started work on Aapconf, but only the first few steps have been taken.



The documentation was written in SGML docbook. The other formats are generated from this.



Makes you wonder why software developers don't have tails...



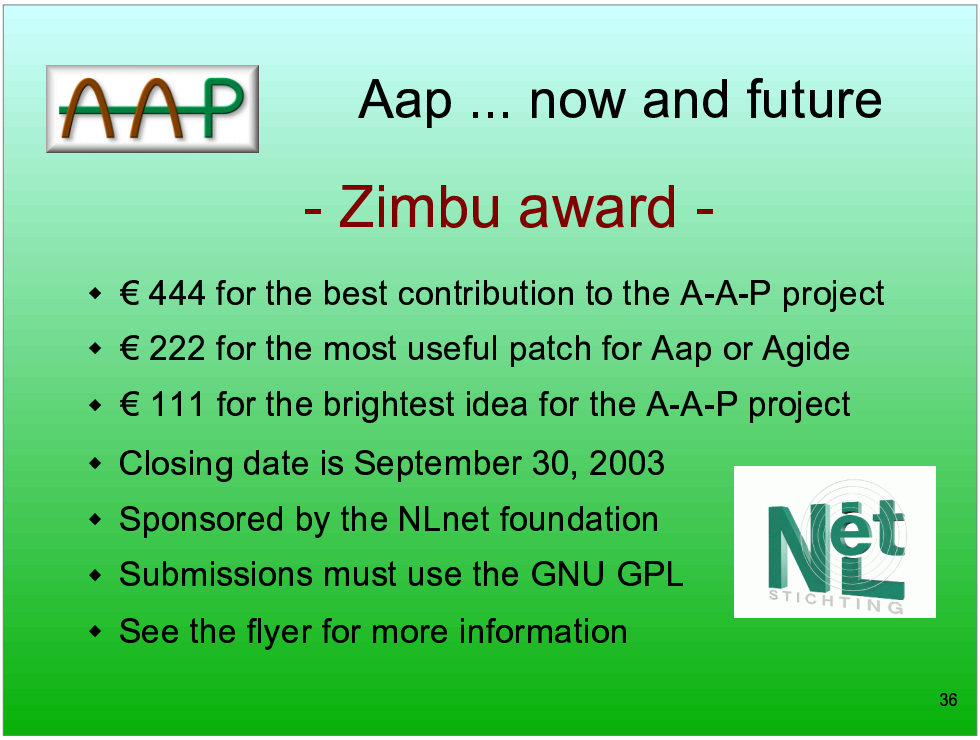
You can download and use Version 1.0 of Aap right now. It works. I use it myself for a wide number of tasks, including distribution of Aap and maintaining several web sites.

For downloading see http://www.a-a-p.org/download.html.

Hopefully more languages and compilers will be supported in the next version. This depends on people to submit their Aap tool plugins.

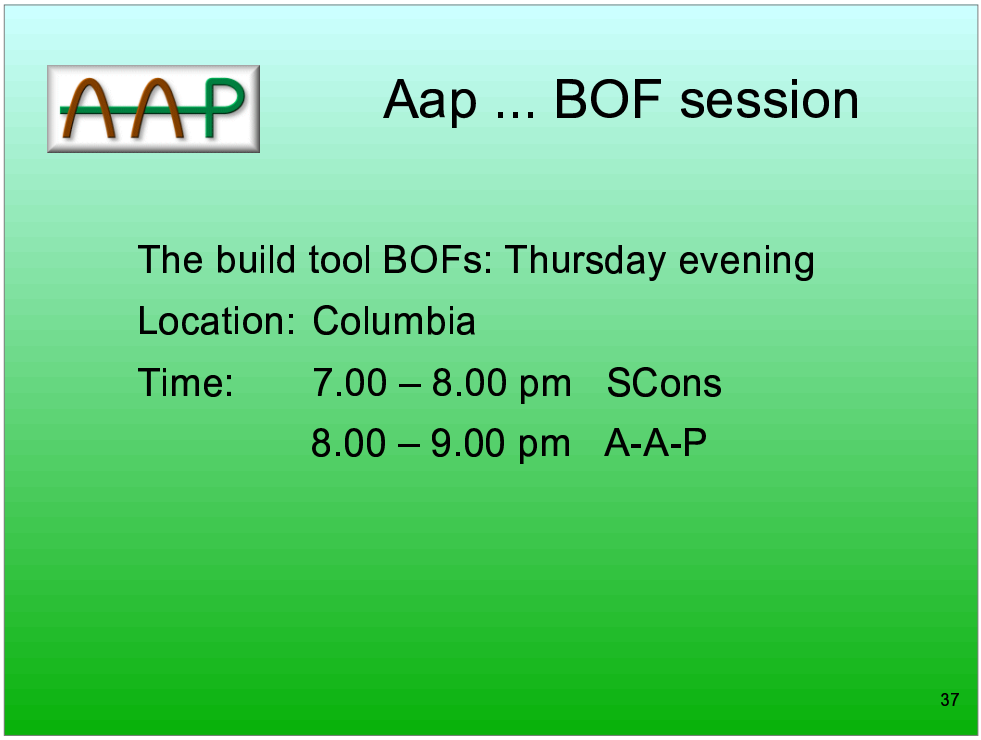
Agide is also available. As mentioned earlier, it can be used for debugging Vim. Not that you would need to do that...

What Aap 2.0 will be is not at all clear. It largely depends on the feedback I receive. Possibly it will include the configuration functionality. Another thing often asked for is a system for distributing software (deployment with proper dependency handling and upgrade support).



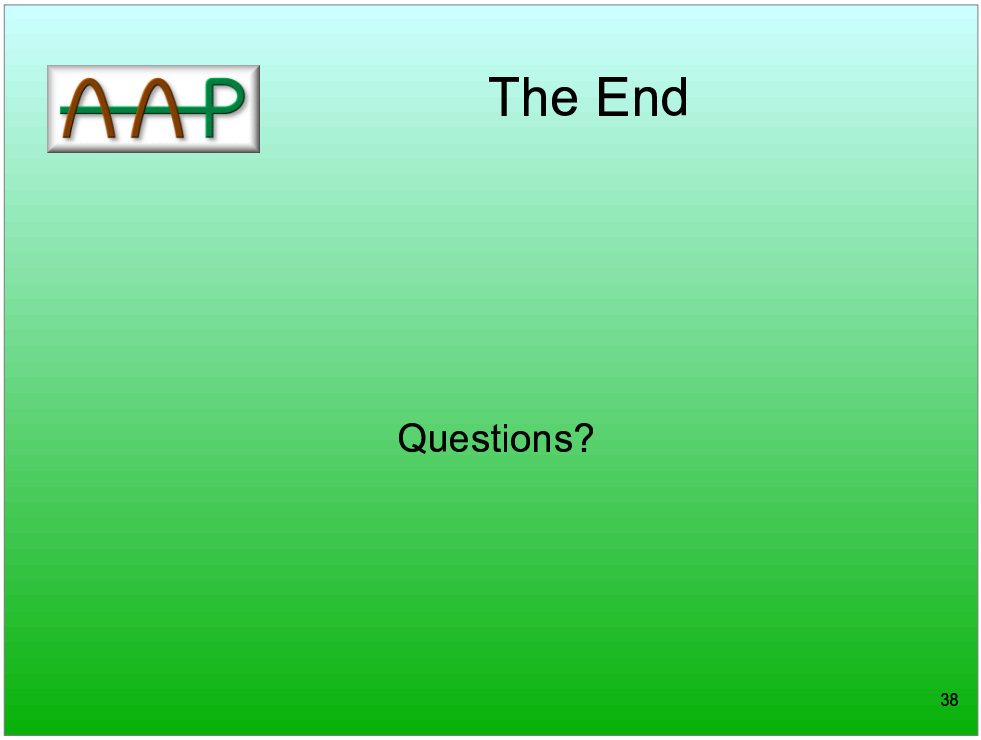
Aap version 1.0 has been released. The next version should be even more powerful, faster, reliable and simpler to use. If you help making this possible you will not only receive appreciation from Aap users, but have a chance to make money as well!

- The “best contribution” can be anything that helps the progress of the A-A-P project or improves the usability of Aap and/or Agide.
- The “most useful patch”can be for fixing a long standing bug, adding support for a specific compiler, a port to another system, etc.
- The “brightest idea” is a suggestion for a useful addition to the A-A-P project or for improving Aap or Agide without the need for an actual implementation.
- Contributions are to be sent to the A-A-P-develop maillist. No specific format is required. Everybody contributing to A-A-P automatically becomes a potential award winner.
- You may submit as many times as you like. All contributions must go under the GNU GPL.
- The closing date is September 30, 2003. The winners will be announced on the A-A-P web site in October 2003.
- The selection of the award winners will be done by Bram Moolenaar, the A-A-P project leader.
- The money is provided by the NLnet foundation.



Tonight will be the perfect time for talking about build tools. The first hour is for SCons, another interesting Python based build facility. The second hour is for A-A-P.

I would like to invite everybody who has suggestions for Aap or wants to ask more questions than what we have time for now. Be there tonight!



This is the end of the presentation.