Micro Guide for HP Work Stations in the SL/AP Group

H. Grote and F. C. Iselin

March 13, 1995

1 Introduction

1.1 Aim

This guide is intended for HP work station users of the SL/AP Group. Most of the guide will be valid for other users (and even other Unix-based work stations), but no effort will be made to point out the idiosyncrasies. The guide is meant to provide, together with an on-line demonstration by the authors, most of the knowledge a typical AP Group member will need to perform the work at hand, i.e.

- create and access data files (lattices, orbit measurements, ...)
- use existing tools to manipulate and view text and data files (editing with emacs, displaying with gnuplot, viewing with ghostview ...)
- prepare data for, and execute existing programs such as ABCI, MAD, WIGWAM, ...
- write own programs in Fortran or C or C++, debug them ...
- use Unix tools to manipulate and structure files, control disk space, find files of which only the name or some string inside the file is known (this usage rises steeply with age, alas), print files, send files to other computers, run remotely on other computers, ...
- write simple command shells, in particular Makefile
- and, last but not least, do not disturb others.

1.2 What is UNIX?

For a complete definition please consult a textbook. Basically, it is a portable operating system from the distant past, the days of teletypes (some kind of line-oriented terminal without screen, but noisy), which led to abbreviated commands (rm for remove, ls for list, cp for copy, cat for concatenate, etc.), and crammed command lines, e.g.

```
> sed 's/xxxx.x/gxplot.a/\n  astap.lbr > astap.lib
```

but is otherwise rather modern in that it allows for parallel execution, pipelines, directory trees, and pretty powerful command languages. A list of all commands can be found in /bin (there is other stuff, too). An on-line help facility explains their usage. Very important for the practical usage: Unix is case sensitive.

1
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>give a different name to a command or a group of commands</td>
</tr>
<tr>
<td>ar</td>
<td>archive (actually used to make binary libraries)</td>
</tr>
<tr>
<td>cc, c89</td>
<td>C compiler</td>
</tr>
<tr>
<td>cat</td>
<td>concatenate - can actually be used to list a file</td>
</tr>
<tr>
<td>cd</td>
<td>change directory (modified in our private installation)</td>
</tr>
<tr>
<td>chgrp, chmod, chown</td>
<td>change file group, access mode, or owner</td>
</tr>
<tr>
<td>cmp</td>
<td>compare two files</td>
</tr>
<tr>
<td>compress</td>
<td>compress a file (can save a lot of space)</td>
</tr>
<tr>
<td>cp</td>
<td>copy a file</td>
</tr>
<tr>
<td>crypt</td>
<td>encode/decode (encryption) files</td>
</tr>
<tr>
<td>csh</td>
<td>set c-shell environment (that’s what we use normally)</td>
</tr>
<tr>
<td>date</td>
<td>print date and time</td>
</tr>
<tr>
<td>diff</td>
<td>compare files or directories</td>
</tr>
<tr>
<td>du</td>
<td>list the disk usage in blocks of 512 bytes</td>
</tr>
<tr>
<td>echo</td>
<td>prints its argument</td>
</tr>
<tr>
<td>ed</td>
<td>edit (line oriented text editor)</td>
</tr>
<tr>
<td>elm</td>
<td>electronic mail interface</td>
</tr>
<tr>
<td>elmalias</td>
<td>make alias list for elm</td>
</tr>
<tr>
<td>env</td>
<td>list environment variables</td>
</tr>
<tr>
<td>exit</td>
<td>from a window or remsh</td>
</tr>
<tr>
<td>find</td>
<td>find a file, or string in file(s) (powerful)</td>
</tr>
<tr>
<td>fsplit</td>
<td>split a file with several Fortran routines into files per routine</td>
</tr>
<tr>
<td>ftp</td>
<td>file transfer</td>
</tr>
<tr>
<td>f77</td>
<td>Fortran compiler</td>
</tr>
<tr>
<td>grep, egrep, fgrep</td>
<td>find text strings in file(s)</td>
</tr>
<tr>
<td>gzip</td>
<td>gnu compress program</td>
</tr>
<tr>
<td>gunzip</td>
<td>gnu uncompress program</td>
</tr>
<tr>
<td>head</td>
<td>list top of file</td>
</tr>
<tr>
<td>history</td>
<td>shows the last 20 commands (or any number you care)</td>
</tr>
<tr>
<td>hostname</td>
<td>print name of current computer (host)</td>
</tr>
<tr>
<td>kill</td>
<td>kill a running process</td>
</tr>
<tr>
<td>ksh</td>
<td>use the standard (restricted) command shell</td>
</tr>
<tr>
<td>l, ls, ll</td>
<td>list files</td>
</tr>
<tr>
<td>ld</td>
<td>link editor (normally performed by compiler)</td>
</tr>
<tr>
<td>ln</td>
<td>link files and directories</td>
</tr>
<tr>
<td>login</td>
<td>get HP documentation</td>
</tr>
<tr>
<td>lrom</td>
<td></td>
</tr>
</tbody>
</table>
3 Environment

3.1 Logging In

When a user logs into the computer, several command files (shell scripts) are executed in which the working environment is defined. Unfortunately, there is no such thing as one single environment.

The simplest working mode is from a dumb terminal, line oriented, e.g. with remote login through another computer. In this case (amongst others, about which we normally know nothing, not even their location) the file .login is executed.

When logging into HP VUE, the full-blown multi-screen multi-window bells-and-whistles interface of these work stations, .login is not executed, but a whole host of files is. The user's environment is set by executing .vueprofile, and the system opens some windows in a predefined layout. The first time the user logs in, the configuration comes from the system file /usr/vue/config/sys.session. For the layout of later sessions refer to Section 4.1.

3.2 Directory Structure in Our Work Stations

To help you with finding files we list in Table 3 the most important directories found on the SL/AP work stations.

3.3 Opening a Window or Running a Shell

Whenever the user opens a new window or runs a command shell, the file .cshrc (C-shell something) is executed. This has the following consequences: when the user wants to define variables that are only interesting for the HP VUE interface, the place is in .vueprofile — no need to define them more than once, usually. Command aliases, library definitions, and file search paths are better put into .cshrc. In any case, if the user suspects that the correct shell start-up has not been executed, the command

source shell_name

e.g.

source .cshrc

will execute the shell without making a shell — otherwise one might end up in an infinite loop, e.g. when executing

.cshrc

which creates a shell which creates a shell which creates a shell ...

3.4 Variables

There are three types of variables, environment variables, shell variables, and aliases.
3.4.1 Environment variables

These variables are global, i.e. once defined they can be accessed everywhere. They depend on the shell type (csh, ksh, sh, tsh, bash ... ) — just forget this for now, but remember it when something suddenly does not work the way you expect it to work. These variables are normally in CAPITAL letters (convention). Some of these variables are defined for all users, such as HOME, the full name of the user's home directory. The default for undefined variables is an empty string. Variables are defined by — in the csh environment —

```csh
setenv NAME value
```

where value must be included in double quotes if it contains blanks. They are referenced by preceding them with a $-sign:

```csh
echo $NAME
```

will print the string given in value. There is a special case where the variable has to be included in brackets as well, i.e. `$NAME` (see Section 10.3). These variables can be referenced inside C-programs. The user can define any number for his convenience. Example: if too lazy to type `/users/fci/mad/mad8.s` each time I want to access this file, I could (through a command) define

```csh
setenv MS /users/fci/mad/mad8.s
```

and then use it freely, e.g.

```csh
grep -i plplot $MS
```

The variable PATH will be explained here in more detail. When a command is issued which does not specify the full directory path, i.e. rather `cd` than `/bin/cd`, then the system looks into the directories specified in PATH, in the order given. No PATH, no command. So if your PATH is empty, or your current directory ./ is not part of it, and you have just compiled a program and proudly type a.out, then the system will say: not found! You would have to specify ./a.out.

We equip all users with a standard path which comprises the directories which they would normally want to use. Its format is obvious, so it can easily be modified.

Table 4 gives a list of important environment variables the user has to know and (sometimes) to modify. Most of them are pre-defined in our standard environment.

3.4.2 Shell variables

They are local to each shell and can therefore be different in different shells. They are defined with

```csh
set name=expression
```

e.g.

```csh
set history=20 # previous commands to remember.
set savehist=0 # number to save across sessions.
set system='hostname' # name of this system.
set prompt = "$system \!: " # command prompt.
```

These variables must of course be defined in .cshrc.
3.5 Opening a remote window

There are two actions needed:

- Tell the target computer you want to open a display there (this is normally your own work station or XTerminal) with:

  \texttt{xhost \textit{hostname}}

  where \textit{hostname} is the name of the source computer (where you create the window).

- Change the \texttt{DISPLAY} variable on the source computer (where you are currently running) with:

  \texttt{setenv \texttt{DISPLAY \textit{hostname:0.0}}}

Any window you now create (with \texttt{xwindow}, or \texttt{emacs}, or \texttt{ghostview}, \texttt{gnuplot} etc.) will then appear on the target computer, as long as \texttt{DISPLAY} is not changed. Example: Suppose user hansg is logged into parcb and wants to use emacs from his work station hpiago. In a local window on hpiago, he types

\texttt{xhost parcb.cern.ch}

and in the parcb window he types

\texttt{setenv \texttt{DISPLAY hpiago.cern.ch:0.0}}
\texttt{emacs &}

and he is in business.

4 HP VUE (HP Visual User Environment)

4.1 Screen Configurations

The screen configuration at log in can be changed using the \textit{style manager} (see Section 13.2). One may set a \textit{home session}, and make it the log in configuration. All information for the home session is stored in files in the directory "\texttt{/vue/sessions/home}.

The user can also resume the configuration existing at log out time. If the log out is completed correctly, this information is placed in the directory "\texttt{/vue/sessions/current}.

4.2 System Crash

After a system crash the current session may have been corrupted, and correct log in is no longer possible. Thus we recommend that you set a home session using the style manager. After a crash you can still log in with the option \textit{failsafe}, delete the whole directory "\texttt{/vue/sessions/current} and log in again using the home session. If you have not set a home session, the last fall-back is to delete the current session as above and to log in again. The system default session will then be resumed.
and paste and for text searching. It also contains a powerful spell checker. Selection Spell and Buffer starts a process which walks through the current buffer, checks every single word, and offers alternatives for misspelt words. Within the spell checker, help is available by typing a question mark. The Help menu provides information about the use of emacs.

Emacs keystrokes can be learned easily: Select the option emacs Tutorial in the Help menu to enter the tutorial. Then read the text displayed, and execute the proposed actions to teach yourself the keystrokes.

5.2 Editing Modes in emacs

For many file types (based on the file name's postfix) emacs adapts its mode to allow easier manipulation of the file text. Many modes have sub-modes which permit further customisation. The most common emacs modes are:

**text mode:** The default for unknown postfix provides simple word wrap: When you type text without new-lines it breaks lines at blanks such as to fill the lines up to the default fill column 72. Moving the cursor to any column and typing Ctrl-X f will set the fill-column to that column position.

**FORTRAN mode:** postfix .f, has no word wrap, but allows automatic indentation by tabbing to column 7. Control structures like IF/THEN/ELSE/ENDIF are further indented by three columns for each nesting level.

**C mode:** postfix .c (C program), or .h (C header), has no word wrap, but indents control structures automatically.

There are many more modes, two of which are mentioned below.

To open an emacs window containing a text file you have several options:

- In a terminal window, type emacs filename.
- In a file manager window, double-click on the file.
- Select the file in a file manager window, then select Open or (for some file types) Edit Text in the Action menu.

In an already open emacs window, you can select Open File ... in the File menu. This prompts you to type in the file name in the bottom line of the window.

5.3 Printing Text Files

To print a text file use one of the following:

- In a terminal window type lp filename. The file must not be a .ps file, unless you want to print the actual postscript code which is rather lengthy. To print files with lines longer than 80 characters, you may use lp -oBSDwnnn filename where nnn is the largest line length in characters. Adding the option -d printer sends the print-out to a printer different from the default.
- Select the file in a file manager window, then select the action Print from the Action menu.
- From a file manager window drag the file to the printer icon on the front panel.

13
8 Making Simple Plots

Simple plots of tables can be built using the gnuint program. This program is an X windows interface to gnuplot. Type gnuint to enter the gnuint input dialog window. This contains the following fields:

**Data File:** The data file to be plotted. Use the *Select* button to select the file in the directory tree.

**Destination:** The name to be given to any postscript file written. Use the *Select* button to name the file and its place in the directory tree.

**Diagram title:** Type the title to be printed on top of the plot.

**Style:** Select the plot style for the next line(s).

**Format:** Select portrait or landscape orientation for printing and ps file. Selecting *EPS file* causes the button *PS File* to write an encapsulated postscript file.

**Curve label:** Type the label for the next curve.

**Selection for x:** The horizontal axis for the plot.

**Selection for y:** The vertical axis for the plot.

**Selection for z:** If selected, the plot is 3-D.

**Set Range:** Select these button to set the plot ranges.

**Clear:** Start a new plot frame.

**Plot:** Add curve to current plot, using the settings made.

**Replot:** Redraw the current frame.

**Print:** Send the current plot to the printer.

**PS File:** Write a postscript file, using the name under *Destination*.

**Quit:** Exit gnuint program.

9 Running MAD

MAD is normally run in batch mode by a command like

```
mad < data.mad
```

where *data.mad* is the name of the data file. A specific version can be selected by one of the options *-old, -pro, or -new*.

In batch mode the plots are written to a .ps file. Running interactively is also possible by the command mad, again with one of the options to select a version. Then you may load a data file by the command call "*data.mad*". In interactive mode you will be asked if you want to display your plots on the screen, and whether you want to write a .ps file containing all plots or one .eps file for each plot.
10.2 Making binary libraries

There are basically two types:

Shared libraries that get loaded only once and can be shared by several programs running simultaneously. They are not part of the binary executable file (which therefore is shorter), but get only loaded at execution.

Archive libraries which are loaded into the executable file. They will be described here.

An archive library has postfix .a and is created from object modules. Now each object module (i.e. file xxx.o) goes as one chunk into the library, and if one of its routines is called, the whole lot is loaded. This means if you compile the whole of the CERN library from one Fortran file into one object file which you convert to a library, then calling one routine loads them all. This is normally not what one wants to achieve. Consequently, you have to make an object module for each Fortran routine. For this, the command fsplit comes handy. So the best way to proceed is this:

- Make a temporary directory, say tmp
- Move your big Fortran file there: mv myfort.f tmp
- Go into tmp: cd tmp
- Split the file: fsplit myfort.f
- Move myfort.f back: mv myfort.f ..
- Compile all files in the directory: f77 -c *.f
- Make an archive library from all object modules: ar rv mylib.a *.o
- Move the library one level up: mv mylib.a ..
- Move up your pwd: cd ..
- delete the temporary directory with all its files: /bin/rm -R tmp

10.3 Makefile

A special program make uses the file with name Makefile as input and is extremely useful, certainly one of the most used features of Unix altogether. The principle is as follows: one or several files are being made that depend normally on other files which in turn may be made from others etc. Dependencies have to be spelt out explicitly. make will only proceed if any of the files necessary for the creation of a file has been changed more recently than itself, otherwise it will say that the file is up-to-date. A trivial example: suppose your Makefile contains the lines

myprog: myfort.f
    f77 myfort.f -o myprog $(LIBS)

(please note the brackets around LIBS). When you issue once
make myprog
argument (if any) given with the command is checked (the arguments are called $1, $2 etc.), and, with the help of a shell variable called `var` links to different MAD directory files are established, and different binary files are executed. The execution is indirect, through the `nice` program which lowers the CPU priority of the program. This has the effect that the interactive terminal response to other users is not reduced considerably. Except for the `-sitf` version, and prior to execution, the file `higz.windows.dat` is copied into the current directory from somewhere if it does not already exist; this file is used by the Higz plot interface to define the plot window size. Simple, ain’t it?

12  A List of Rules and Tools

12.1 Background Execution

Whenever you execute a command in a window, the window is blocked until the command has finished: no big deal when you have many windows at your disposal; however, if logged in through one window only, it is useful to know that (any) execution can be relegated to background execution by adding an ampersand "&" behind it, e.g.

```
mad < mydata.mad > myfile.out &
```

This will create a process with a number, and will then let you continue to type into the window. When the process has finished, you will get a message, however only after the first command typed in after it finished, so don’t wait for this message, the time may get long. The process number is useful when you want to stop the process with `kill`.

12.2 Nice

Another important feature is running with `nice`. We request that this be used whenever you are not the only user on a workstation or server, and execute something taking more than a few seconds. In that case, please precede your command with

```
nice +10
```

which has the effect of lowering your CPU priority from the default level. It means in practice that the response time for interactive usage (typing into a window, for example) does not degrade because of your job running, whereas the execution time of that job will hardly be affected. Incidentally, `nice +10` is already built into the MAD execution shell `mad`.

12.3 Disk Space and Backup

No system ever has enough disk space, this is a well known fact. On our HP servers, a user can reasonably use up to 70 MBytes before he will cause attention (and possibly action). The `compress` or `gzip` facilities allow to make sometimes considerable savings (up to a factor seven). To find out about your disk space, use `du`.

Once per week (Friday at 22:05 h), all user files on the HP system are backed up to tape. These tapes are normally recycled after three month, but the last tape of each month is kept indefinitely. Up to now there has been no disk crash on any of our HP stations, but this is no guarantee for the future. If you have an essential file that has taken many hours of your time to make, please copy it onto another disk.
# ghostview action

ACTION GhostView

  COMMAND
  TYPE
  WINDOW-TYPE NO-STDIO
  EXEC-STRING /usr/local/bin/X11/ghostview \
  \%(File)Arg_1"File To View:"% 
  DESCRIPTION The GhostView action runs the ghostview \ 
  interpreter to view a .ps file.

END

where the following fields have been used:

#: Comment line.

TYPE Normally COMMAND, indicating that a program is to be run.

WINDOW-TYPE TERMINAL causes a terminal window to open which will be closed when the 
program exits. PERM-TERMINAL, causes a terminal window to open which remains open when the program exits. NO-STDIO tells the system that no terminal window is needed.

EXEC-STRING The command to be executed. The backslash at the end of the line indicates continuation. When the action is invoked on a file, the string 
\%(File)Arg_1"File To View:"% 

is replaced by the name of that file. When it is invoked by double-clicking on the action itself, the system displays a dialog box to prompt for the file name with the 
prompt File To View:

DESCRIPTION The help text for Help - On Item on the action itself.

4. Open the General tool box, double-click on System_Admin and the double-click on Reload_Actions.

After these three steps the action will be accessible from your personal tool box, and from the file manager.

13.4 Modifying the Front Panel

You can modify the layout of the front panel by editing the file ~/.vue/vuewmrc. The file is well structured, and the changes to be made are not too difficult to see. To make your changes active, you have to log out and log in again.

13.5 Modifying X Resources

Programs running under the X windows system are controlled by so-called resources which include things like colours, fonts, button labels, window sizes etc. The easiest way to modify resources is to use the action EditResources in the general tool box, in subdirectory System_Admin. However, you should do this only if you know what you do. The main problem is to know the resource names. Usually they can be found in the documentation for the programs.