abduco & dvtm
Session and Tiling Window Management for the Console

Marc André Tanner

CoSin ’18
Oberon

Alpha Oberon

Copyright (C) 1991-1995 by ModulaWare, France

Alpha Oberon is an implementation of Oberon for Digital Equipment Corporation AlphaAXP OpenVMS workstation (OSF/Motif) or AlphaAXP OpenVMS server with Xenix.

Both the programming language Oberon2 and the Oberon System have been implemented. A complete description of the language and of the system, one should read the following:

Tutorial for the Oberon programming language and concise language reference.

User manual for the programming environment and reference for the standard module library.

N. Wirth and J. Gutknecht: Project Oberon. The Design of an Operating System and Compiler.
Program listings with explanations for the whole Oberon system, including the compiler for NS32000.

Oberon V4

H. Möszenböck

Oberon V4 is a cleaned up version of Oberon V2.2 (The version number V3 refers to “Gadgets Oberon” which has a graphical user interface.) It resulted from the desire to integrate the text editors Edit and Write. This document explains the differences between V4 and V2.2. What is not described here remains as explained in the book “The Oberon System” by M. Reiser (see reference [1] below).
NAME
acme, win, awd - interactive text windows

SYNOPSIS
       win [command]
       awd [label]

DESCRIPTION
Acme manages windows of text that may be edited interactively or by external programs. The interactive interface uses the keyboard and mouse; external programs use a set of files served by acme; these are discussed in acme(4).

Any named files are read into acme windows before acme accepts input. With the -l option, the state of the entire system is loaded from loadfile, which should have been created by a Dump command (q.v.), and subsequent file names are ignored. Plain files display as text; directories display as columned lists of the names of their components, as in ls -l directory, except that the names of subdirectories have a slash appended.

The -f (-F) option sets the main font, usually variable-pitch (alternate, usually fixed-pitch); the default is /lib/font/bit/lucidasans.euro.8.font
(/usr/lucida/unicode.e.9font). Tab intervals are set to the width of 4 (or the value of $Tabstop) numeral zeros in the appropriate font.

Windows
Acme windows are in two parts: a one-line tag above a multi-line body. The body typically contains an image of a file, as in sam(1), or the output of a program, as in r(1) window. The tag contains a number of blank-separated words, followed by a vertical bar character, followed by anything. The first word is the name of the window, typically the name of the associated file or directory, and the other words are commands available in that window. Any text may be added after the bar; examples are strings to search for or commands to execute in that window. Changes to the text left of the bar will be ignored, unless the result is to change the name of the window.
X11: wmi, wmii, dwm
TL;DR: What is this all about?

Provide a similar working environment suitable for:

- Framebuffer console
- Remote e.g. SSH/mosh sessions
TL;DR: What is this all about?

Provide a similar working environment suitable for:
- Framebuffer console
- Remote e.g. SSH/mosh sessions

abduco
- session persistence
TL;DR: What is this all about?

Provide a similar working environment suitable for:
► Framebuffer console
► Remote e.g. SSH/mosh sessions

abduco
► session persistence

dvtm
► tiling window management for the console
► terminal multiplexer
Tiling window management

- Optimally use available screen space
- No overlapping windows
- Automatic window placement
- Minimal window decorations
Hi,

For some time I have been thinking about applying the concept of tiling window management to the console. As a result I have written dvtm, you can check it out here:

http://www.brain-dump.org/projects/dvtm/

...
Concepts shared with dwm

- window management should be automatic and dynamic
- master and stacking area
- tagging concept
- similar key bindings, MOD defaults to Ctrl-g
- 1-line statusbar (via a named pipe)
- configuration through config.def.h
Design Philosophy

Heavily influenced by suckless.org.

Focus on simplicity, clarity and frugality, minimal but useable, do one thing and do it well.

- *dynamic* window management for the console
- no internal copy mode (use `$EDITOR` instead)
- no session support (see abduco)
- easily scriptable
dvtm – *dynamic* virtual terminal manager

Single *modifier* key, prefix for all commands

Denoted by $\text{	exttt{MOD}}$, defaults to $\langle \text{C-g} \rangle$
dvtm – *dynamic* virtual terminal manager

Single *modifier* key, prefix for all commands

Denoted by $MOD$, defaults to $\langle C-g \rangle$

Can be changed at runtime:

- dvtm -m ^a ($\langle C-a \rangle$ as in screen)
- dvtm -m ^b ($\langle C-b \rangle$ as in tmux)

Use $MOD-$MOD to send the $MOD$ key.
dvtm – window lifecycle

- dvtm process1 process2 ...
- $MOD-c$ create new window
- $MOD-C$ create new window with same working directory\(^1\)
- $MOD-x-x$ close window

Closing the last window, terminates dvtm.

\(^1\) Depends on /proc/$PID/cwd
dvtm – focus windows

- $\text{MOD-}j$ focus next
- $\text{MOD-}k$ focus previous
- $\text{MOD-}J$ focus next non-minimized
- $\text{MOD-}K$ focus previous non-minimized
- $\text{MOD-}[0..9]$ focus $n$-th window
- $\text{MOD-}\langle\text{Tab}\rangle$ focus previously selected window
dvtm – master and stacking area

Available space is split into two areas:

- master: the primary window(s)
- stacking: the other windows
dvtm – change master area

▸ $\text{MOD-}〈\text{Enter}〉$ swap current window to/from master area
dvtm – change master area

- $\text{MOD-}\langle\text{Enter}\rangle$ swap current window to/from master area
- $\text{MOD-1}$ increase master area width
- $\text{MOD-h}$ decrease master area width
dvtm – change master area

- $MOD-⟨Enter⟩$ swap current window to/from master area
- $MOD-1$ increase master area width
- $MOD-h$ decrease master area width
- $MOD-i$ increase number of windows in master area
- $MOD-d$ decrease number of windows in master area
dvtm – minimize/maximize windows

▶ $\text{MOD}-.$ toggle minimization of current window
▶ $\text{MOD}-m$ maximize current window
dvtm – layouts

A way to place/display windows.

$\text{MOD}-\langle\text{Space}\rangle$ cycles through layouts.
dvtm – layouts

A way to place/display windows.

$\text{MOD-}\langle \text{Space}\rangle$ cycles through layouts.

- $\text{MOD-}f$ Vertical stack (default)
- $\text{MOD-}b$ Bottom stack
- $\text{MOD-}g$ Grid
- $\text{MOD-}m$ Monocle/fullscreen
dvtm – layouts

A way to place/display windows.

$\text{MOD}–\langle\text{Space}\rangle$ cycles through layouts.

- $\text{MOD}-f$ Vertical stack (default)
- $\text{MOD}-b$ Bottom stack
- $\text{MOD}-g$ Grid
- $\text{MOD}-m$ Monocle/fullscreen

Also included in source tarball, but disabled by default:

- Top stack
- Vertical stack
- Fibonacci: spiral & dwindle
dvtm – tagging concept

Controls which windows are displayed.

A super set of the workspace functionality.
dvtm – tagging concept

Controls which windows are displayed.

A super set of the workspace functionality.

A static set of tags = \{tag1, tag2, \ldots, tagN\}

Every window is tagged with at least one tag.
dvtm – tagging concept

Controls which windows are displayed.

A super set of the workspace functionality.

A static set of $tags = \{tag_1, tag_2, \ldots, tag_N\}$

Every window is tagged with at least one tag.

A view is a subset of $tags$ i.e. ($views \subseteq tags$)

A view displays all windows having at least one of the tags.
dvtm – tagging modifying the view

View tag: display all windows with tag\textit{N}, "change workspace"

µ $\text{MOD-v-}N$
View tag: display all windows with \textit{tagN}, "change workspace"

▶ $\text{MOD-v-N}$

Toggle tag of view: add/remove all windows with \textit{tagN}

▶ $\text{MOD-V-N}$
Tag window: apply $tagN$ to focused window, "move window to workspace"

▶ $\text{MOD-t-N}$
Tag window: apply \textit{tag}N to focused window, "move window to workspace"
\begin{itemize}
  \item[$\text{MOD}-t-N$]
\end{itemize}

Toggle tag of window: add/remove \textit{tag}N from focused window
\begin{itemize}
  \item[$\text{MOD}-T-N$]
\end{itemize}
dvtm – miscellaneous tagging

$\text{MOD}–v–\langle \text{Tab} \rangle$ switch to previously selected tags

$\text{MOD}–0$ view all tags / windows
dvtm – status bar

Hidden by default.

Displays a single line of text, read from a FIFO:

```bash
▶ mkfifo -m 600 dvtm.status
▶ dvtm -s dvtm.status
▶ echo "your fancy status" > dvtm.status
```

See `dvtm-status(1)` for an extended example

$MOD-s
toggles status bar

$MOD-S
cycles position (top, bottom)
dvtm – status bar

Hidden by default.

Displays a single line of text, read from a FIFO:

▶ mkfifo -m 600 dvtm.status
▶ dvtm -s dvtm.status
▶ echo "your fancy status" > dvtm.status

See dvtm-status(1) for an extended example
dvtm – status bar

Hidden by default.

Displays a single line of text, read from a FIFO:

▶ mkfifo -m 600 dvtm.status
▶ dvtm -s dvtm.status
▶ echo "your fancy status" > dvtm.status

See dvtm-status(1) for an extended example

$MOD-s toggles status bar

$MOD-S cycles position (top, bottom)
dvtm – scrollback buffer

Enhances terminals like st(1) with a scroll back buffer.

- Set history size: `dvtm -h lines`
- `<S-PageUp>` or `$MOD+<PageUp>` scroll up
- `<S-PageDown>` or `$MOD+<PageDown>` scroll down
dvtm – keyboard multiplexing

Keypresses are forwarded to all visible windows.

Useful to interactively manage multiple servers.

- $MOD-a$ toggles multiplexing mode
dvtm – copymode

Copy and paste text across windows.

- uses your $EDITOR as interactive filter
  - pipes scroll back buffer history to editor
  - keeps whatever the editor writes to stdout in a register
  - dvtm-editor(1) makes it work for ordinary $EDITORs
dvtm – copymode

Copy and paste text across windows.

- uses your $EDITOR as interactive filter
  - pipes scroll back buffer history to editor
  - keeps whatever the editor writes to stdout in a register
  - dvtm-editor(1) makes it work for ordinary $EDITORS

- $MOD-e enter copy mode
- $MOD-p paste previously copied text
dvtm – window title

Xterm terminal escape sequence extension:

$ printf "\033]0;%s\007" "Your title here!"

See also dvtm-title(1)
Titlebar (or tagbar) indication that "something" occurred in the window.

Triggered by ASCII bell character \a.
dvtm – mouse support

Click to focus window.

Double click to focus and toggle maximization.

Middle click to zoom.

Right click to minimize.
dvtm – scripting capabilities

Control dvtm from other processes.

Reads commands from a named pipe.
dvtm – scripting capabilities

Control dvtm from other processes.

Reads commands from a named pipe.

- `dvtm -c dvtm-command.fifo`
- `echo "create vis" > dvtm-command.fifo`
- `$DVTM_CMD_FIFO` exposed to child processes
dvtm – scripting capabilities

Control dvtm from other processes.

Reads commands from a named pipe.

- `dvtm -c dvtm-command.fifo`
- `echo "create vis" > dvtm-command.fifo`
- `$DVTM_CMD_FIFO` exposed to child processes

Only unidirectional communication.

Still limited and experimental.
abduco: session handling

- Provides session persistence
  - terminate stuck SSH sessions \( \langle \text{Enter} \rangle \sim . \)
  - `ssh user@host -t abduco -A session`
abduco: session handling

- Provides session persistence
  - terminate stuck SSH sessions \( \langle \text{Enter} \rangle \sim . \)
  - ssh user@host -t abduco -A session

- Simple client/server architecture
- Communication over Unix domain socket
abduco: session handling

- Provides session persistence
  - terminate stuck SSH sessions \( \langle \text{Enter} \rangle \sim . \)
  - `ssh user@host -t abduco -A session`

- Simple client/server architecture
- Communication over Unix domain socket

- Operates on the raw I/O stream
- Does not attempt to interpret or preserve terminal state
abduco: basic usage

Create session (and attach)

▶ abduco -c demo
abduco: basic usage

Create session (and attach)
▶ abduco -c demo

Detach session
▶ ⟨Ctrl-\⟩
abduco: basic usage

Create session (and attach)
  ➤ abduco -c demo

Detach session
  ➤ ⟨Ctrl-\⟩

Reattach session
  ➤ abduco -a demo
<table>
<thead>
<tr>
<th>Status</th>
<th>Last Activity</th>
<th>Server PID</th>
<th>Session Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Sat 2018-06-16 20:40:36</td>
<td>27492</td>
<td>connected</td>
</tr>
<tr>
<td></td>
<td>Sat 2018-06-16 20:39:49</td>
<td>27414</td>
<td>inactive</td>
</tr>
<tr>
<td>+</td>
<td>Sat 2018-06-16 20:40:13</td>
<td>27487</td>
<td>dead</td>
</tr>
</tbody>
</table>
$ abduco

Active sessions (on host thinkpad)
* Sat 2018-06-16 20:40:36 27492 connected
  Sat 2018-06-16 20:39:49 27414 inactive
+ Sat 2018-06-16 20:40:13 27487 dead

Column meaning:
1. Status, * active / client connected, + terminated
2. Last activity (mtime of socket)
3. Server PID
4. Session name
abduco: session exit status

No output buffering, but exit status is recorded.

$ abduco -n demo false && abduco -a demo
donduco: demo: session terminated
with exit status 1
Multiple simultaneously connected clients.
  ▶ Most recently non-readonly client dictates pty size
  ▶ Read only sessions (input is discarded)
  ▶ For security purposes, use socat(1)

$ socat -u unix-connect:/tmp/abduco/private/session
   unix-listen:/tmp/abduco/public/read-only &
abduco: resize handling

Most recently non-readonly client dictates `pty(7)` size.

Delivers `SIGWINCH` to underlying process.
abduco: socket recreation

In case session socket disappears:

- `pgrep -P 1 abduco`
- `lsof -p $PID | grep unix`
- `kill -USR1 $PID`
- `cp /proc/$PID/exe abduco`
- `./abduco`
abduco: environment variables

Command to run, if omitted:
- $ABDUCO_CMD defaults to dvtm

Current session information:
- $ABDUCO_SESSION
- $ABDUCO_SOCKET
Limitations & a plan to fix them

Terminal state not preserved across sessions

Possible fix:

1. session attached
2. abduco sends signal to supervised application (i.e. dvtm)
3. dvtm restores terminal state
Future Plans

- Find more time for maintenance
- Preserve terminal state across sessions
- Improve terminal emulation
  - 24 bit color support
  - $dvtm \approx dwm + st$?
  - $dvtm \approx libvterm + libtickit$?

$^2$In no particular order, no timeline given.
Future Plans

- Improve scripting capabilities, allow bidirectional communication via a unix domain socket
  
  $ echo cmd | socat - UNIX-CONNECT:/tmp/socket | doit

- Provide Lua API?

- Resolve abduco license controversy
Conclusion

Does not conflate session and window management.

Although raw edges, conceptually sound.

Non-bloated solution which works (at least for my usecase).
Questions?

https://github.com/martanne/abduco
https://github.com/martanne/dvtm

git://repo.or.cz/abduco.git
git://repo.or.cz/dvtm.git

mat@brain-dump.org

#vis-editor on freenode

Happy Hacking!