TAGS: acme plan9 git

Acme Tricks

2021, December 11

Contents: Acme for `git`; using it as your terminal; fixing `zsh`; changing the font; running on WSL; leveraging `plumber`; using the new Go port; finding information; getting started

References: Plan9 Port; Cat-v; UNIX HATERS Handbook (esp. § *6. Terminal Insanity*)

I was a die-hard *emacs* user for a long time. In 2014 I discovered *Acme*, and though these days I do much of my work on *VS Code* and *GoLand*, *Acme* remains my Swiss Army knife.

My knowledge grows in fits and starts, driven by need. Information on *Acme* is not so easy to find. I thought I'd share a few tricks I've gathered.
Intended for intermediate users, but I have some information for beginners at the end.

Acme as a "porcelain"[1] for git

In my new job I've had some scary moments with `git rebase`, which I had not really used in previous jobs. A couple of colleagues had to help me out. On one occasion I needed to `squash` a long list of commits. With `vim` serving as the default `EDITOR` for my interactive rebase, my colleague walked me through going into *visual block mode*, choosing a bunch of `pick` commands strewn over many lines, and changing them to `s`. Whew!

I realized I needed to become better at `vim`, or perhaps start using `emacs` again to leverage `magit`. But these kinds of thoughts annoy me. All I should need to do is a) understand rebasing, and b) have a good way to edit rebase lists. Having to learn visual block mode or use `magit` or switch to a GUI tool seemed totally overkill for b).

1. Using Acme as your EDITOR

`git` gives you hooks for your favorite tools. The environment `EDITOR` variable will tell git which program to use for "visual" operations like interactive rebasing or writing commit messages. In most systems it defaults to `vi` or `vim`.

In my personal computer I used to set `EDITOR` to `sam -d` to use *sam* in line-editor mode, much like `ed`. This gave me a retro kick but it was too clunky for rebasing.

Plan9 Port comes with two very useful scripts: `B` and `E`. `B` opens content in a new window inside your *Acme* session. `E` does the same, but returns to the calling process only after you're done editing. This is perfect for `git`.

You can set `EDITOR=E` in your shell config file of choice, or you can use it on an as-needed basis, e.g. `EDITOR=E git rebase -i HEAD~5`

NOTE: you should have a `plumber` process running for this to work.[2] More info on the plumber below.

2. Editing rebase lists

With your `EDITOR` pointing to `E`, now you can do an interactive rebase.

```
/Users/jsilvela/m/proyectos/mediary/.git/rebase-merge/git-rebase-todo
pick 7351bcd Fix bug where text accumulated across diary entries
pick 977c74b Fix file creation/appending code
pick 4a3e899 Clean up new diary file creation logs.
pick e3b6e66 Silly sample update
pick 79f5df1 Cannon fodder

# Rebase dbb03e8..79f5df1 onto dbb03e8 (5 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
```

Say you want to squash a bunch of commits. You select the relevant lines, then execute `Edit x/pick/ c/squash/` in the tag. No need for new modes or plugins.
When you're done editing the rebase manifest, remember to `Put` and `Del` to give the calling `git` the go-ahead.

3. Use Acme to show git commits

A nice thing in *Acme* is how easy it is to execute commands anywhere by middle-clicking. When working with `git`, sometimes you'll want to do a `git log`, perhaps a `git log --oneline`. It would be great if you could right-click on a commit SHA, and get a new window open with the results of `git show`.

```
/Users/jsilvela/m/proyectos/mediary/ Del Snarf Get | Look git log --oneline
.git/              .vscode/          diag.dot          go.sum
.gitignore         LICENSE.txt       diag.svg          mediary.go
.idea/             README.md         go.mod            mediary_test.go
/Users/jsilvela/m/proyectos/mediary/+Errors Del Snarf | Look
79f5df1 Cannon fodder
e3b6e66 Silly sample update
4a3e899 Clean up new diary file creation logs.
977c74b Fix file creation/appending code
7351bcd Fix bug where text accumulated across diary entries
dbb03e8 Use diary's readers/writers, add unit test
58cd297 Update license
3c4da9b Added license
4d66f36 Lint and fmt
```

For this, we need to use the *plumber* (more on it later). You should `pgrep` to check if you have a `plumber` process. If you don't, fire it.[2] And perhaps think of adding it to your `.zlogin`, `.profile` or equivalent.

You can add the following rule[3] to your `~/lib/plumbing` file:

```
type    is      text
data    matches '[0-9a-f]*[a-f][0-9a-f]*'
data    matches '([0-9a-f][0-9a-f][0-9a-f][0-9a-f][0-9a-f][0-9a-f]+)'
plumb   start rc -c 'cd '$wdir'; cd `{git rev-parse --show-toplevel} &&
```

That's a handful. Let's analyze it:

1. The `matches` statements ensure we only apply the rule on SHA-1 looking strings. The parentheses tell us what we're reading into the $1 value.
2. `rc -c`: we're passing our complex instructions to the `rc` shell as a string, as we need to go beyond plumbing rule syntax.
3. `cd `{git rev-parse --show-toplevel} && [-d .git] && git show '$1'`: make sure we're on a `git` repo, then call `git show` on the SHA-1 looking argument we found previously.
4. `| plumb -i -d edit`: pipe the output of `git show` to an edit (Acme) window. The `-i` argument tells `plumb` to read the input from STDIN (in our case piped), and the `-d` specifies the *destination*.
5. `-a ''action=showdata filename=+git/'$1'''` is not strictly necessary, but it will add a title to the window tag showing the commit SHA.

The plumber and 9p

The composition model in Unix is that of programs writing and reading standard I/O, connectable by pipes.

The designers of *Plan9* wanted an even more general and reusable mechanism: the composition model in *Plan 9* was that of file servers. Even Acme is written as a file server.

It provides access to its resources, including its windows, via the *9p* protocol.

For instance, you can list the windows open in Acme:

```
9p ls acme
```

for which I get currently

```
12
2
24
44
acme
cons
consctl
draw
editout
index
label
log
new
```

Those numbers are window id's. You can search inside them with `9p ls` and reveal subdirectories. Right now, running `9p read acme/12/body` will display the contents of the window where I'm writing this post.

The *plumber* is the Plan9 program that connects programs. You can think of it as the next iteration of Unix pipes in the richer environment of file servers.

Plan9 Port comes loaded with some rules for the plumber. For example, it contains rules to send url's to a web browser or image files to an image viewer, so entering `plumb http://wikipedia.org` or `plumb myDocument.pdf` in your terminal will open stuff in other programs.

Remember the `plumber` process needs to be running.[2] By default it daemonizes, but you can add the `-f` flag to keep it in the foreground while you debug your rules.

You can define your own rules in `~/lib/plumbing`. In the Plan9 Port installation you can find predefined rules in `$PLAN9/plumb/`. The man pages on the plumber are reasonably good, once you know where you're going.

Acme as your terminal

One (more) peculiarity about Plan 9 and Acme: they do away with all the Terminal Emulation crap.[4] Many Plan9 programs will list "dumb" as their terminal type. Way to go.

This is important to keep in mind when running a `Win` terminal in Acme. It's a very different experience from what you're used to: this is still a **text** window. You can edit everything, even history, with full mouse support. Any text in the window can be highlighted and executed via the middle button, and will be added to the terminal session. Being able to edit a long CLI invocation with the mouse seems totally natural, but is sadly lacking in (most?) terminal emulators.

However, some Unix/Linux commands are wired to use terminal emulation and pagers, and won't work too well on Acme `Win`. You'll get a fair warning:

```
WARNING: terminal is not fully functional
-  (press RETURN)
```

This will happen, for example, if you try to do a `git diff` inside a repo folder. You could get around this by setting the `PAGER` environment var to `p`, a pager that comes installed with Plan9 Port and does not rely on terminal emulation.
Another solution is to the use plumber once again, to pipe the output of git to a new window:

```
git diff | plumb -i -d edit
```

zsh on Acme

`zsh`, which is the default on *macOS* and ubiquitous these days, shows a strange character sequence on the prompt, in Win:

```
Mac-Mini-2 ~ % [?2004h
```

You can easily google or stack-overflow this to find that the `^[[?2004h` and `^[[?2004l` control sequences turn *bracketed paste mode* on/off. This mode doesn't play well with "dumb" terminals.

You can deactivate it with `unset zle_bracketed_paste`

Acme on Windows WSL

Arguably, running Debian on WSL2 on a Windows machine may be a better Unix experience than macOS. Strange times!

It's not difficult to get Acme running smoothly on WSL2. You can find a nice general discussion in this donationcoder thread.

What you need:

1. Get yourself an X client. For example I got x410.
2. Set the `DISPLAY` environment variable to specify the location of the X server.

In WSL1, `export DISPLAY=:0` worked. In WSL2, you should use the nameserver IP address created by WSL.

This works:

```
export DISPLAY=$(cat /etc/resolv.conf | grep nameserver | awk '{print $2
```

You can add it to your login shell startup script.

The Go port of Acme

This hasn't been advertised much, and I only heard about it by chance in the recent interview Dev Tool Time with Roger Peppé.

You can find the new port of Acme written in Go in the 9fans go repo. The editor is in the `cmd/acme` directory, not the top-level `acme` folder (which contains utils).

This is not the first effort to port Acme to Go, but being driven by Russ Cox, this one feels as official as it could be. It's already working decently.

To build and use, clone the repo and `cd` into it, then

```
go build -o acmePort 9fans.net/go/cmd/acme
```

and move the executable (however you've named it) to some place in your PATH.

You can also run directly with `go`:

```
go run 9fans.net/go/cmd/acme
```

Changing the font

There's no config file for Acme, and no config menus. You set the font via CLI arguments.

The fonts will be served to Acme through `fontsrv`, which is installed with Plan9 Port. To find out the filenames of available fonts: `fontsrv -p .` (don't forget the dot.) The filenames are long, so it's a good idea to save the one you want to use as an environment variable in your shell startup.

For example I use the font Input, set

```
export FONT=/mnt/font/InputMonoNarrow-Regular/20a/font
```

in my `.zlogin`, and then simply call:

```
acme -a -f $FONT &
```

Motivation, getting started, finding information

The best way to see if learning Acme might interest you is to watch Russ Cox's video A Tour of the Acme Editor.

One of the principles of Acme and other Plan9 programs was to take advantage of the mouse. This is in sharp contrast with the keyboard-heavy ethos of `vi` or `emacs`. The evidence on mouse-driven vs. keyboard-driven is not clear[5], but personally I'll say that remembering so many commands and so many modes in emacs/vi gets tiring. Acme has fewer but more powerful primitives.

The Plan9 Port page has all the information you need to get the system installed and running.

A large part of the power of Acme comes from the command language, which is inherited from *sam*. Rob Pike has a great guide on the sam command language.

Example recipe: `,x g/Emacs/p` will go through the file and print any line that contains the string `Emacs`. That comes from Pike's *sam* tutorial. To execute this in Acme, you need to run it with an `Edit`. The converse of the `g//p` idiom to print matching lines would be `v//d` to delete non matching lines.

A few useful recipes, then:

- Add an extra line break between lines: `Edit x/\n/ c/\n\n/`
- Print lines matching `foo`: `Edit ,x g/foo/p`
- Delete lines not matching `foo`: `Edit ,x v/foo/d`

Acme and the other Plan9 Port utilities are not widely used. This makes it more difficult to find information about them, but it's also satisfying to be so removed from the stack-overflow instant-answer industrial complex.

The plan9port-dev mailing list is not really the place for beginner questions, but the searchable archives are very useful, and the people are helpful.

There's also this directory: mailing lists.

For background, place in the world, and a good rant, I like Cat-v. Good rants are underrated.

---

1. yes, really, *porcelain* is the term that gets used for git UI. Oh well. ↩
2. `plumber`, as well as `plumb`, are installed as part of Plan9 Port. If you added `$PLAN9/bin` to your PATH, you should see them. `plumber` requires no arguments, and daemonizes. ↩ ↩ ↩
3. lifted verbatim from user *squeek* in a discussion on plan9port-dev ↩
4. The UNIX HATERS Handbook has much to say here that I agree with (see § *6. Terminal Insanity*) – though I'm no Unix Hater. ↩
5. Dan Luu has a post on keyboard vs mouse. ↩

Email GitHub Mastodon