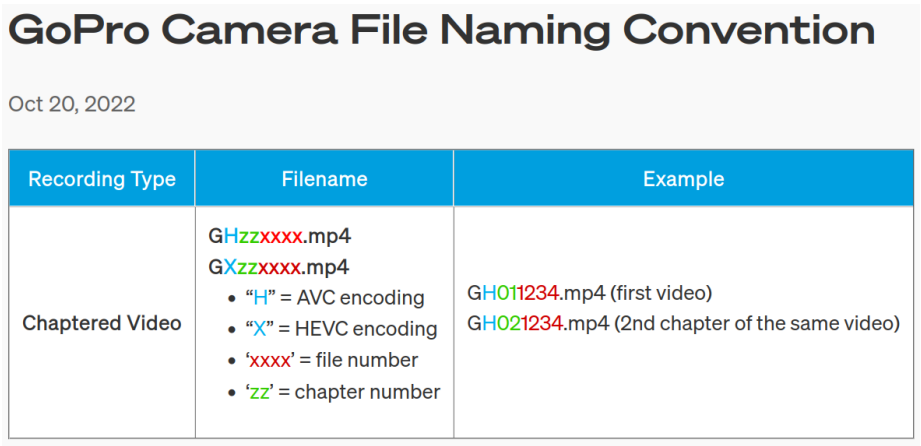
Shell for Fun and Profit: Handout

Dieses Handout fasst den Vortrag *Shell for Fun and Profit* zusammen und ist zusammen mit den Demos zu genießen.

Motivation

Es gibt Tage, da will man einfach in die Tastatur beißen: Man hat ein paar Stündchen Videomaterial mit der GoPro gedreht, doch wegen dem verfluchten Naming Scheme sind bei alphanumerischer Sortierung die Datein durcheinander.



GoPro Camera File Naming Convention

Oct 20, 2022

Recording Type	Filename	Example
Chaptered Video	GHzzxxxx.mp4 GXzzxxxx.mp4 • "H" = AVC encoding • "X" = HEVC encoding • 'xxxx' = file number • 'zz' = chapter number	GH011234.mp4 (first video) GH021234.mp4 (2nd chapter of the same video)

Figure 1: Bescheuerte Naming Conventions, die nach geskriptetem Umbenennen schreien.

Oder man will mal vielleicht schnell ein paar Zahlen summieren, sortieren, den Mittelwert bilden oder so, aber Excel macht leider Schlapp, weil es ein paar mehr Zahlen sind.

DevOps Borat
@DEVOPS_BORAT

Big Data is any thing which is crash Excel.

6:25 PM · Jan 8, 2013

1,877 Retweets **21** Quotes **660** Likes **2** Bookmarks

Figure 2: Multi-Gigabyte CSVs, die Excel ins Schwitzen bringen.

Oder die Testsuite läuft lokal durch, aber im Docker-Container in der Pipeline sagt der Computer *nein*.

Figure 3: Testsuite, die lokal durchläuft, aber in der Pipeline explodiert, ca. 2022, Kolorisiert.

In diesen (und anderen) Situationen kann das kleine Fensterchen "Shell" mit seinen Freund:innen und Helfer:innen, den UNIX utilities, das Licht in finsteren Stunden sein:

- Dateien umbenennen mit regex, `for`-Schleife und mutigem `mv`en erspart Stunden des *Rechtsklick*, *Umbenennen*, *Tippen*, *Weinen*.
- Größere Dateien verarbeiten z.B. `sort`, `uniq` und `awk` ohne großes Wimpernzucken.
- Und wenn man sich in der Shell zurechtfindet, dann kommt man wahrscheinlich auch im Docker-Container, fernab der nächsten GUI, klar um mal zu untersuchen, was genau die Pipeline explodieren lässt.

Basics

Hier eine kleine (nicht unbedingt vollständige) Auflistung von Shell/UNIX-Dingen, die ich oft tippe.

- `.`, `..`
- `ls`, `cd`
- `touch`, `mkdir`, `cp`, `rm`, `mv`
- `*`, `?`
- `man`
- `cat`, `head`, `tail`
- `sort`, `uniq`, `grep`
- `>`, `>>`, `<`, `|`, `$( ... )`, `<( ... )`
- `if`, `for`, `||`, `&&`
- `$var`
- `&`
- *strg + r*

Mit so einer Liste kann man vielleicht nicht allzu viel anfangen außer ein *"jo, kenn ich"* bzw. ein *"hä"* in sich hineinzugrummeln. Daher sei hier auf die Key Points: Shell, Git, etc des Buches "Research Engineering in Python" verwiesen. Dort gibt es eine ähnliche Liste, jedoch ergänzt um sinnvollen Text und nebenbei noch ganze Kapitel, die die Konzepte einführen und erklären.

Powertools

Neben dem, was ich so als die "Basics" bzw. das "Panzertape" wahrnehme, gibt es dann noch die Powertools: Die Schlagbohrer, Kettensägen, Nagelfeilen etc.: Werkzeuge für konkrete Aufgaben/Gebiete. Hier mal ein paar, die mir grad so einfallen.

- `vim`: *der* Texteditor. Gibt's auch als VS Code/Intellij Plugin.
- `ranger`: Filemanager mit `vim`-keys
- `find`: listet Dateien auf
- `diff`, `vimdiff`: Unterschiede zwischen Dateien
- `make`: simples build system
- `curl`: http tool
- `jq`: json tool
- `pandoc`: Document Converter
- `yt-dlp`: YouTube (etc) Downloader
- `ffmpeg`: Audio/Video tool
- `imagick`: Bildkonverion etc.
- `git`: Versionsverwaltung
- `tmux`: Terminal Multiplexer (Mehrere Fenster in einem)
- `PDFtk`, `PDFjam`, ...: PDF-Tools
- `espeak-ng`: Simple Text-To-Speech Engine
- `notify-send` bzw. `libnotify`: Desktop Notifications

Und

- `R`, `gnuplot`, `matplotlib`: Plotten / Visualisieren
- `php`: Simple Webprogramme in weniger Zeilen Code (vgl. ytp-Demo)

Und

- `shellcheck`: Linter für Shell-Skripte.

Footguns

Wie bei jedem tollen Werkzeug bietet auch die Shell tolle Möglichkeiten der permanenten Pediküre: ValveSoftware / steam-for-linux #3671

Versehentliche Flags

Stellen wir uns mal vor, wir haben folgenden Ordner:

```
$ tree --charset=ascii
.
|-- bar
|-- baz
|-- -f
|-- foo
|-- -r
`-- wichtige_dokumente
    `-- bitcoin_wallet.docx

1 directory, 6 files
```

Wir wollen nun alle Dateien im aktuellen Ordner, also `foo`, `bar`, `baz` und `-r` sowie `-f` löschen. Der Unterordner `wichtige_dokumente` soll erhalten bleiben.

Da Zeit auch Geld ist, entscheiden wir uns für ein flottes `rm *`, das löscht ja dann nur die Dateien und nicht den Unterordner, nicht war? Tja, leider wird `*` zuerst von der Shell zu `rm bar baz -f foo -r wichtige_dokumente` aufgelöst und `rm` interpretiert dann `-r` und `-f` als flags, weshalb wir damit leider unser Wallet gelöscht haben:

```
$ tree --charset=ascii
.
|-- -f
`-- -r

0 directories, 2 files
```

Quoting von Variablen

Hier fehlen beim `rm`-Befehl die Anführungszeichen rund um die Variable, daher werden die falschen Dateien gelöscht:

```
$ touch foo bar "foo bar"
$ ls
 bar   foo  'foo bar'
$ FILE="foo bar"
$ rm $FILE
$ ls
'foo bar'
```

Command schlägt fehl, Skript macht weiter

Stellen wir uns vor, wir haben folgendes Verzeichnis:

```
$ tree --charset=ascii
.
|-- cleanup.sh
`-- folder
    `-- garbage

1 directory, 2 files
```

Wobei die `cleanup.sh` wie folgt aussieht:

```
$ cat cleanup.sh
cd fodler
rm *
```

Nun haben wir uns leider verschrieben (`fodler` statt `folder`), weswegen `cd` fehlschlägt und `rm` daher in `.` anstelle von `folder` ausgeführt wird. Aber hey, immerhin löscht sich das kaputte cleanup-Skript selbst...

Die Moral von den Geschichten: Shellcheck

Shellcheck findet alle eben genannten Footguns und kann auch als Plugin in die handelsüblichen Editoren integriert werden.

Fun Fact: Die folgende Footgun findet Shellcheck (noch) nicht:

```
COLUMNS=foo
echo $COLUMNS  # prints "foo"
touch bar
echo $COLUMNS  # prints terminal-width
```

Warum Shell/UNIX lernen?

Personal *Computing*

Grad im universitären Kontext wird Programmieren in einer sehr allgemeinen[1] Programmiersprache beigebracht. Als Aufgaben gibt's dann viel Algorithmik und so. Das ergibt natürlich Sinn, das Gelernte ist aber kaum im Alltag anwendbar.

Shell zu lernen ermächtigt Studierende mMn nun, die algorithmischen PS auf die Straße zu bringen, denn die Shell ermöglicht *interaktives* und *effektives* Programmieren von (halbwegs) alltäglichen Problemen.

Wenn man dann in das Linux Rabbithole reingleitet, Stück für Stück zu (Text)dateien-fokussierter Software (z.B. LaTeX oder Markdown/pandoc statt Word, `mutt` statt Thunderbird, moc mit lokaler Musiksammlung statt Spotify, ...), ergeben sich interessante Synergien:

- Alles lässt sich mit `git` Versionsverwalten.
- Alles lässt sich mit `find`, `ripgrep` etc. durchsuchen.
- Alles wird Skriptbar, mit `vim` editierbar, ...

Ganz nebenbei lacht man inzwischen über Webseiten wie jpg2png.com, pdf-merge.com, etc.

Man ist nun nicht mehr nur Konsument:in von Software, sondern auch Produzent:in.

Zinseszins

Shell ist nicht nur alltäglich, sondern tendenziell auch noch lange nützlich, denn Shell – also konkret z.B. `ash`, `dash` oder `bash` und die ganzen tollen Prográmmchen wie `head`, `tail` (z.B. aus den gnu coreutils) – wird's auch in 20, 40, ... Jahren noch geben. Selbst wenn Linux irgendwann nicht mehr das dominante (serverseitige) Betriebssystem ist, gehe ich davon aus, dass der POSIX-Standard (der u.a. `sh`, `head` etc. definiert) uns noch lange erhalten bleibt.

Also selbst wenn es jetzt langsam und mühselig ist, den Bums zu lernen: Man kann Jahrzehnte lang von dem Wissen profitieren.

[1] Im Gegensatz zu einer eher "domänenspezifischen" Sprache wie Shell (Domäne: Interaktive Dateimanipulation), PHP (Domäne: Web), R (Domäne: Statistik).

Right Tool for the Job^{TM}

Hier ein Potpurri an Gedanken über Programmiersprachen etc. als Werkzeuge.

Blub-Paradoxon

Eine stehengebliebene Uhr zeigt zweimal täglich die richtige Uhrzeit, ein blindes Huhn findet mal ein Korn und Paul Graham hatte mit dem *Blub Paradoxon* mal einen interessanten Gedanken:

> I'll begin with a shockingly controversial statement: programming languages vary in power.
>
> ...
>
> Between Perl 4 and Perl 5, lexical closures got added to the language. Most Perl hackers would agree that Perl 5 is more powerful than Perl 4. [O]nce you've admitted that, you've admitted that one high level language can be more powerful than another. And it follows inexorably that, except in special cases, you ought to use the most powerful you can get.
>
> This idea is rarely followed to its conclusion, though. After a certain age, programmers rarely switch languages voluntarily. Whatever language people happen to be used to, they tend to consider just good enough.
>
> Programmers get very attached to their favorite languages, and I don't want to hurt anyone's feelings, so to explain this point I'm going to use a hypothetical language called Blub. Blub falls right in the middle of the abstractness continuum. It is not the most powerful language, but it is more powerful than Cobol or machine language.
>
> And in fact, our hypothetical Blub programmer wouldn't use either of them. Of course he wouldn't program in machine language. That's what compilers are for. And as for Cobol, he doesn't know how anyone can get anything done with it. It doesn't even have x (Blub feature of your choice).
>
> As long as our hypothetical Blub programmer is looking down the power continuum, he knows he's looking down. Languages less powerful than Blub are obviously less powerful, because they're missing some feature he's used to. But when our hypothetical Blub programmer looks in the other direction, up the power continuum, he doesn't realize he's looking up. What he sees are merely weird languages. He probably considers them about equivalent in power to Blub, but with all this other hairy stuff thrown in as well. Blub is good enough for him, because he thinks in Blub.

> When we switch to the point of view of a programmer using any of the languages higher up the power continuum, however, we find that he in turn looks down upon Blub. How can you get anything done in Blub? It doesn't even have y.
>
> [Continues by rambling that Lisp is at the top of the power continuum.]

Knuth vs McIlroy

Zum Lösen folgender Aufgabenstellung

> Read a file of text, determine the n most frequently used words, and print out a sorted list of those words along with their frequencies.

hat Donald Knuth[2] stabile sieben Seiten Text und Code geschrieben. Siehe Abbildung 4 für einen Auszug.

A LITERATE PROGRAM

Last month's column introduced Don Knuth's style of "Literate Programming" and his WEB *system for building programs that are works of literature. This column presents a literate program by Knuth (its origins are sketched in last month's column) and, as befits literature, a review. So without further ado, here is Knuth's program, retypeset in* Communications *style.* —*Jon Bentley*

Common Words Section

1. Introduction. The purpose of this program is to solve the following problem posed by Jon Bentley:

> Given a text file and an integer k, print the k most common words in the file (and the number of their occurrences) in decreasing frequency.

Jon intentionally left the problem somewhat vague, but he stated that "a user should be able to find the 100 most frequent words in a twenty-page technical paper (roughly a 50K byte file) without undue emotional trauma."

frequency; or there might not even be as many as k words. Let's be more precise: The most common words are to be printed in order of decreasing frequency, with words of equal frequency listed in alphabetic order. Printing should stop after k words have been output, if more than k words are present.

2. The *input* file is assumed to contain the given text. If it begins with a positive decimal number (preceded by optional blanks), that number will be the value of k; otherwise we shall assume that $k = 100$. Answers will be sent to the *output* file.

```
define default_k = 100 {use this value if k isn't
        otherwise specified}
```

3. Besides solving the given problem, this program is supposed to be an example of the WEB system, for people who know some Pascal but who have never seen WEB before. Here is an outline of the program to be constructed:

```
program common_words (input, output);
  type ⟨Type declarations 17⟩
  var ⟨Global variables 4⟩
  ⟨Procedures for initialization 5⟩
  ⟨Procedures for input and output 9⟩
  ⟨Procedures for data manipulation 20⟩
  begin ⟨The main program 8⟩;
  end.
```

Figure 4: Auszug aus Knuth's Lösung

Die Lösung wurde zusammen mit einem Review von Douglas McIlroy[3] veröf-

[2] Ja, der hier.
[3] Jo, auch er ist fame.

fentlicht. Er lobt vieles und erkennt an, dass Knuth ja eigentlich nur Literate Programming demonstrieren wollte, lässt sich am Ende aber doch zu einem Roast hinreißen:

> Knuth has shown us here how to program intelligibly, but not wisely. I buy the discipline. I do not buy the result. He has fashioned a sort of industrial-strength Faberge egg - intricate, wonderfully worked, refined beyond all ordinary desires, a museum piece from the start.

Denn er bevorzugt eine einfachere Lösung:

```
tr -cs A-Za-z '\n' |
tr A-Z a-z |
sort |
uniq -c |
sort -rn  |
sed  ${1}q
```

Während die Prägnanz von McIlroy's UNIX-Lösung natürlich insbesonderer im Vergleich zu einem *literate program* beeindruckt, darf man nicht vergessen, McIlroy ein wenig an Knuth vorbeiredet bzw. Knuth eine Falle gestellt wurde.

Knuth will sein System demonstrieren und als Pate der Informatik lässt er es sich natürlich nicht nehmen, ordentlich Algorithmen und Datenstrukturen zu lehren. McIlroy will ebenso sein System demonstrieren und kann es daher nicht lassen, (lehrreiches) overengineering zu monieren. Die Lektion hier: Intention und Kontext sind wichtig bei der Wahl der Werkzeuge.

Der ACM Artikel , in dem sich das ganze abspielt ist btw. sehr nett zu lesen.

Werkzeuge

Nach der Geschichtsstunde eben hier mal ein paar verschiedene Arten zu programmieren:

Tool	Dev Env	Umwelt
Excel	Tabelle	Tabelle
"Progra-like" Java	IDE	Aufgabe
Unix	Shell	Dateisystem, OS
Jupyter	Notebook	CSV, Plots
Backend Webdev	IDE, Postman	REST, ORM/SQL
C / C++ / Rust	IDE, perf	Hardware

Ich denk', dass es hilfreich ist, diese (und andere) Ansätze und ihre Stärken und Schwächen zumindest grob einschätzen zu können, denn traurig sind die, die versuchen...

- ... schnelle Software in Python zu schreiben.

- ... eben mal schnell einen Prototypen in C++ zu schreiben.
- ... ihre cleanup-Skripts in Java schreiben.

Wie es im Tao geschrieben steht:

> Each [tool] has its purpose, however humble. Each [tool] expresses the Yin and Yang of software. Each [tool] has its place within the Tao.

Shell lernen

Meiner Meinung nach kann man Shell/UNIX super durch's Benutzen lernen, z.B. beim Hosten einer Nextcloud oder bei täglichen Rumschlagen mit Linux.

Nextcloud o.ä. Self-hosten

Home

0. Internetzugang mit IPv4
1. Hardware: RasPi / alter Laptop
2. Ubuntu LTS installieren
3. `ssh` einrichten
4. *Port forwarding* am Router einstellen

Cloud

3. (Überspringe die vorherigen Schritte)
4. Mini-Instanz bei Hetzner, NetCup (Sale!), ...

Setup

5. DynDNS-Dienst raussuchen, einrichten
6. TLS-Zertifikat von Let's Encrypt holen
7. Nextcloud o.ä. nach Rezept installieren

Linux als Daily Driver

1. Laptop beschaffen:
   - Alten Laptop rumliegen haben
   - Gebrauchten Business-Laptop holen *(z.B. Dell Latitude, Lenovo X240 / T440)*
   - All-in gehen
   - Dual boot
2. Distro raussuchen *(z.B. Ubuntu, Manjaro)*
3. Benutzen
   - Customizen
   - Scripten
   - ricing (c.f. /r/unixporn)
   - Informieren: YouTube, Reddit, ...

Mein Setup bis neulich

Distro	Manjaro
window manager	i3
Editor	`vim`
File manager	`ranger`
Music Player	Erst `moc`, dann `mpd` mit `ncmcpp`
Video Player	mpv
PDF reader	zathura
Mailprogramm	`mutt`
Todos	`todoman` mit vdirsyncer
Password Manager	`pass`
Moodle Syncer	syncMyMoodle

An der TU inzwischen natürlich isia bzw. isisdl.

Resourcen

Hier die wichtigen Dinge.

- Key Points: Shell, Git, ... Die Liste bietet einen super Überlick über Shell (und nebenbei noch Git und anderes. Ich denke das Buch müsste einen super Einstieg in die im Studium eher wenig gelehrten Aspekte des *Engineering* sein.
- ChatGPT
- The Missing Semester. *"Classes teach you all about advanced topics within CS, from operating systems to machine learning, but there's one critical subject that's rarely covered, and is instead left to students to figure out on their own: proficiency with their tools. We'll teach you how to master the command-line, use a powerful text editor, use fancy features of version control systems, and much more!"*
- Software Carpentry Tutorial: Shell. Ein sehr entspanntes, gutes Shell-Tutorial.
- The AWK Programming Language Eins der besten Bücher über eine Programmiersprache, die ich bisher gelesen hab. Zumindest das erste Kapitel (18 Seiten) sollte man lesen.
- AT&T Archives: The UNIX Operating System Ein tolles kleines Filmchen aus den 80ers (oder so), das das damals brandneue UNIX vorstellt.