

Shell Programming

Different shells (e.g. bash, ksh, tcsh, ash, sh) => different commands/scripts

Why a shell script?

- simple way to string together a bunch of UNIX-commands
- scripts are usually fast to get going
- portable across the whole UNIX world

Nevertheless: scripts are controversial.

- syntax is often ambiguous, wrong documented
- interpretation sometimes leads to surprising results

Helpful webpages:

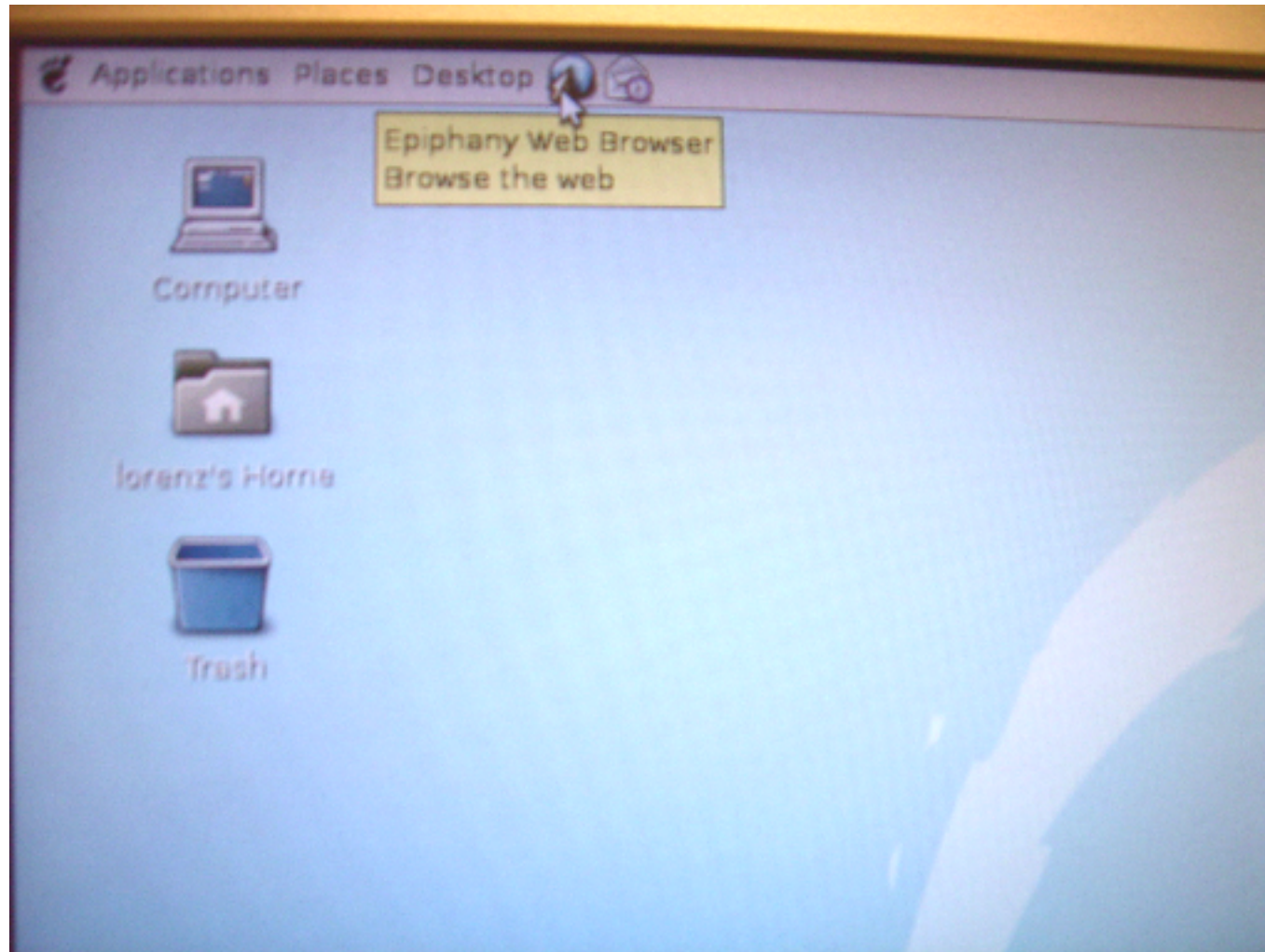
<http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html>

<http://tldp.org/LDP/abs/html/>

<http://www.google.de>

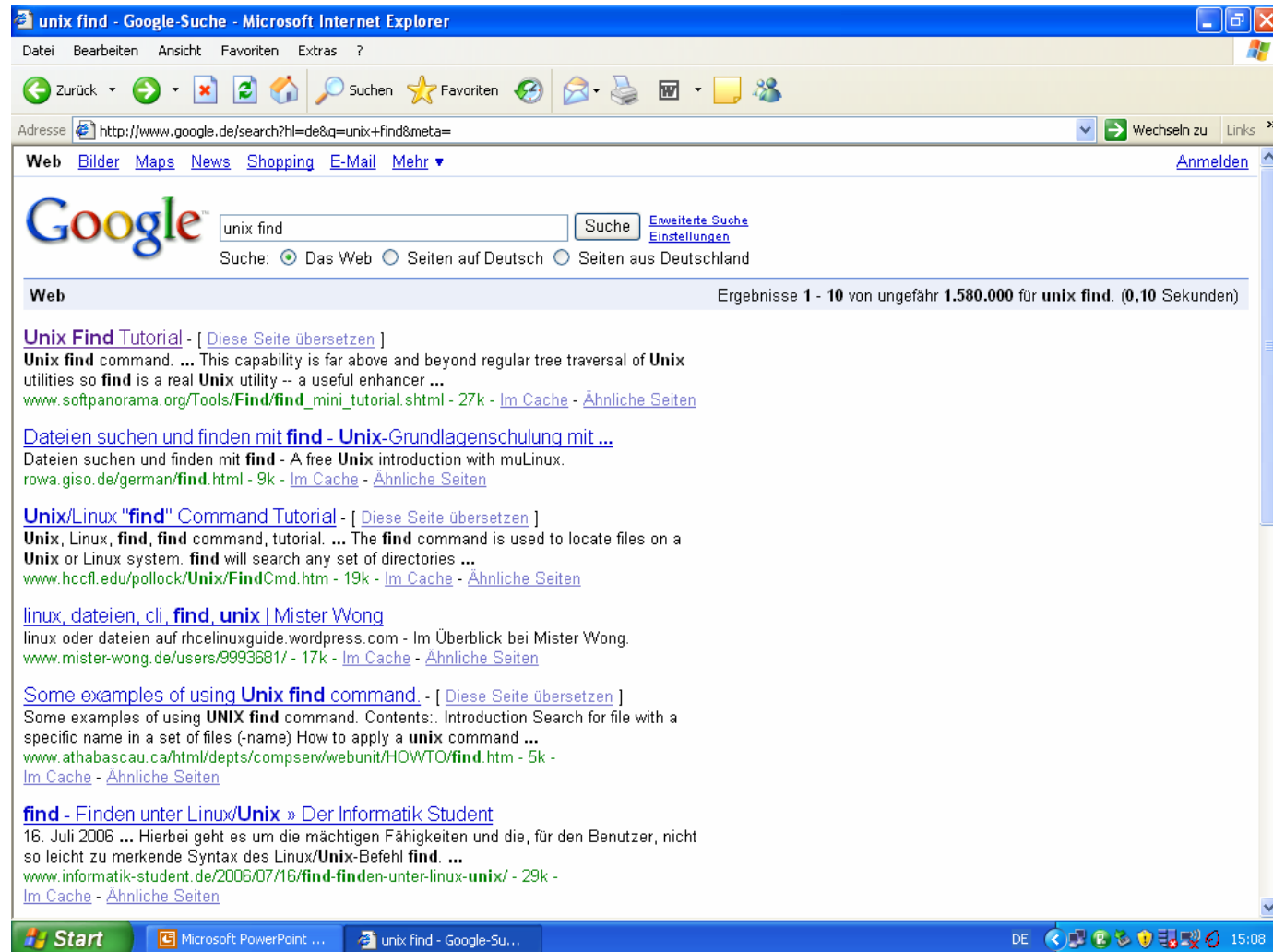
Organisation

- google



Organisation

- google



The screenshot shows a Microsoft Internet Explorer browser window with the title "unix find - Google-Suche - Microsoft Internet Explorer". The address bar contains the URL "http://www.google.de/search?hl=de&q=unix+find&meta=". The search results page displays the Google logo and the search query "unix find". Below the search bar, there are radio buttons for "Suche: Das Web" (selected), "Seiten auf Deutsch", and "Seiten aus Deutschland". The search results are listed under the heading "Web" and show "Ergebnisse 1 - 10 von ungefähr 1.580.000 für unix find. (0,10 Sekunden)". The first result is "Unix Find Tutorial - [Diese Seite übersetzen]" with a description: "Unix find command. ... This capability is far above and beyond regular tree traversal of Unix utilities so find is a real Unix utility -- a useful enhancer ...". Other results include "Dateien suchen und finden mit find - Unix-Grundlagenschulung mit ...", "Unix/Linux 'find' Command Tutorial - [Diese Seite übersetzen]", "linux, dateien, cli, find, unix | Mister Wong", and "Some examples of using Unix find command. - [Diese Seite übersetzen]". The Windows taskbar at the bottom shows the Start button, open applications like "Microsoft PowerPoint ..." and "unix find - Google-Su...", and the system tray with the date "DE" and time "15:08".

Google (part 1): Reading your mind

see <http://www.ams.org/featurecolumn/archive/pagerank.html>



TECHNISCHE
UNIVERSITÄT
DARMSTADT

The problem:

given is

- a library with 25 billion documents
- no centralized organisation
- no librarians
- anyone can add documents

You are interested in information. You only know some keywords.

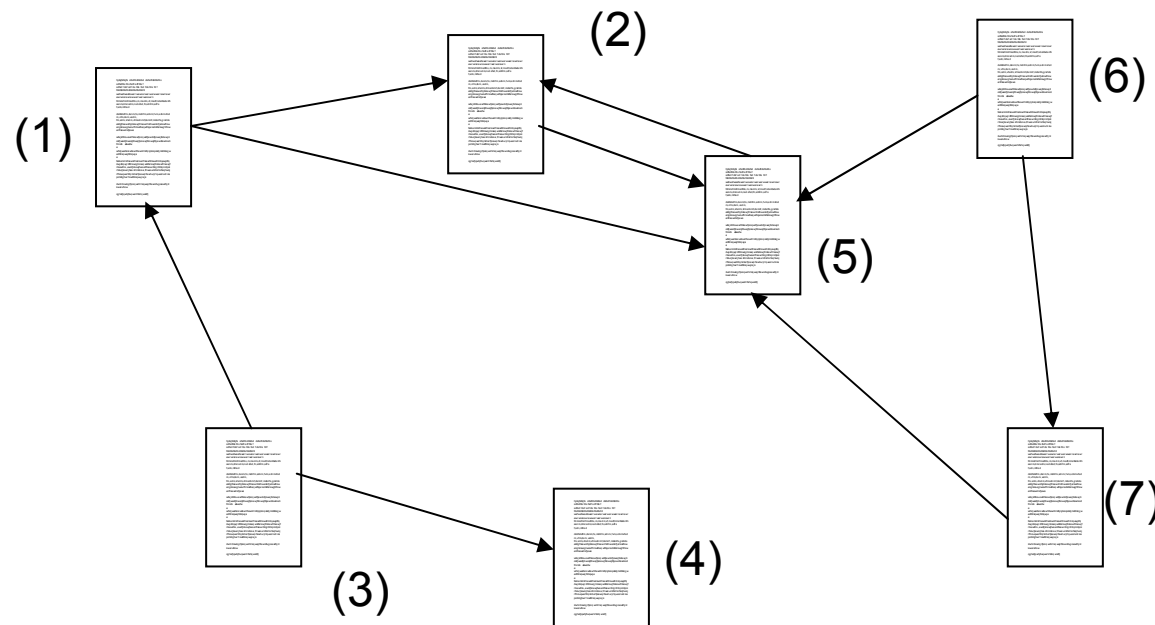
further complication:

Google claims more than 25 billion indexed pages. 95% of the text in the Web is composed of only some 1,000 words. How can we distinguish the important pages from the unimportant ones?

Impossible?

Google (part 1): Reading your mind

The heart of the google software is the PageRank algorithm.



Let P be a web page.
We call $I(P)$ the importance of P .

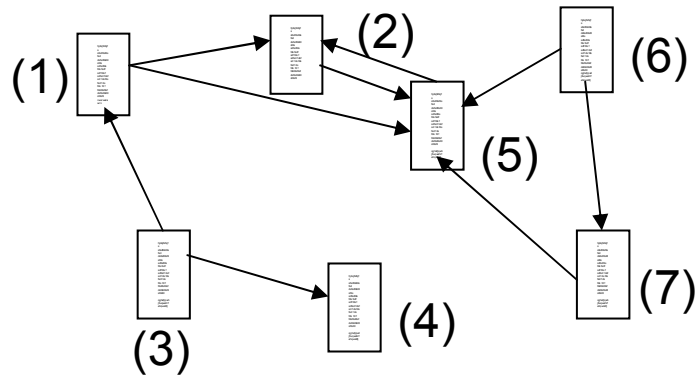
Let P_j have l_j many outgoing links.

If P_i is such a page, P_j will pass $1/l_j$ „importance“ to P_i .

Let B_i be the set of pages linking to P_i . Then the importance relation between a page and its neighbours is as follows:

$$I(P_i) := \sum_{P_j \in B_i} I(P_j) / l_j$$

Google (part 1): Reading your mind



$$I(P_i) := \sum_{P_j \in B_i} I(P_j) / l_j \quad ?? \rightarrow \text{chicken vs. egg problem}$$

Define a matrix $\mathbf{H} = (h_{ij})$ with

$$h_{ij} := \begin{cases} 1/l_j & \text{if } P_j \in B_i \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1 & 0 & 0 & 0 & 0 & 1/2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 \end{pmatrix}$$

and a vector \mathbf{I} of PageRanks: $\mathbf{I} :=$

$$\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Then $\mathbf{I} = \mathbf{H} \cdot \mathbf{I}$.

In other words:

\mathbf{I} is **eigenvector** of the matrix \mathbf{H} with **eigenvalue 1**.

to be continued ... soon

Organisation

- How to get a shell



The bash

A first example

```
#!/bin/bash -vx
echo "My first script" > fst_exa.txt
cat fst_exa.txt
cat fst_exa.txt | wc
rm fst_exa.txt
```

The script has got the name `fst_exa.bash`. The ‚correct‘ way to start is **`/bin/bash fst_exa.bash`**. Shorter: **`./fst_exa.bash`**

Another simple example, using parameters → `def_params.bash`

```
#!/bin/bash
echo "Script name is           [$0]"
echo "First Parameter is       [$1]"
echo "Second Parameter is      [$2]"
echo "Process ID is             [$$]"
echo "Parameter Count is        [$#]"
echo "All Parameters             [$@]"
echo "The FLAGS are              [$-]"
```


The bash

Useful example-commands for scripting are

sort	sorts lines in various orders
grep	searches for expressions in strings or files
basename	strips the path from a path string to leave just the filename
dirname	removes the file from a path string to leave just the pathname
cut	prints selected parts of lines from a file to stdout → cf. <code>unix_for_while.bash</code>
wc	count the characters, words, or lines
tr 'a' 'b'	transform characters → <code>unix_tr.bash</code>
expr	simple arithmetic processor → example in <code>unix_expr.bash</code>
eval	evaluate variables → example in <code>unix_eval.bash</code>
echo	output strings
date	shows a date string
head tail	access head- or tail-lines in files
tar	packing or unpacking an archive (a collection of several files in one file)

The bash

Accents and quotations → `unix_accents_quotes.bash`
(be careful with spaces!)

```
#!/bin/sh
# This is a comment!
echo "Hello World"    # This is a comment, too!
echo "Hello World"
echo "Hello * World"
echo Hello * World
echo Hello World
echo "Hello" World
echo Hello "        " World
echo "Hello \"*\" World"
echo `hello` world
echo 'hello' world
```

The bash

Complex Commands group simple commands into control sets.

- loop structures:

- **for *name* [in ...] do list-of-commands done**

```
alphabet="a b c d e"           # Initialise a string
count=0                        # Initialise a counter
for letter in $alphabet        # Set up a loop control
do                              # Begin the loop
    count=`expr $count + 1`    # Increment the counter
    echo "Letter $count is [$letter]" # Display the result
done                            # End of loop
```

The bash

Complex Commands group simple commands into control sets.

- loop structures:

- **while condition do list-of-commands done**
→ `unix_for_while.bash`

```
alphabet="a b c d e"
count=0
while [ $count -lt 5 ]
do
    count=`expr $count + 1`
    position=`bc $count + $count - 1`
    letter=`echo "$alphabet" | cut -c$position-$position`
    echo "Letter $count is [$letter]"
done
```

Initialise a string
Initialise a counter
Set up a loop control
Begin the loop
Increment the counter
Position of next letter
Get next letter
Display the result

The bash

Complex Commands group simple commands into control sets.

- the if structure
 - **if** *condition1* **then** list-of-commands1
 [**elif** *condition2* **then** list-of-commands2] ...
 [**else** list-of-commandsn] **fi**

Example: → `unix_test_if.bash`

```
#!/bin/bash -vx
if test -w $1
then
    echo "File $1 existiert" #-w file exists and write-rights
fi
```

```
#in the following: [...] implicitly calls "test"
if [ -f $1/$2 ] #-f file exists and is simple file
then
    echo "This filename [$2] exists"
elif [ -d $1 ] #-d directory exists
then
    echo "This dirname [$1] exists"
else
    echo "Neither [$1] nor [$2] exist"
fi
```