JTTE-020003: Topics in Modern Computer Science

Michael Kohlhase

Computer Science
Jacobs University, Bremen Germany
m.kohlhase@jacobs-university.de

April 20, 2016

Preface

The Course

While there are different theories about the impact of technology on human nature and culture we can certainly all agree that we are living in an increasingly tech-heavy age. As global networks become more integrated and active, and the way we interact with texts and documents becomes more computer-supported, students from all academic disciplines will benefit from fundamental concepts and tools for dealing with digital documents and from the ability to think critically about the use(s) of technology.

This course will introduce students to modern document representation, management, and distribution technologies. These technologies are a central – but by far not the only – aspect of Computer Science. But the underlying mechanisms and principles are very much hidden away under the user interfaces that "naive" users use for dealing with documents in their daily lives. This course attempts to reveal some of these underlying mechanisms and forces for a non-CS audience and along the way the course expose students to basic topics in Computer Science.

This Document

This document contains the course notes for the Triangle Course "Topics in Modern Computer Science" held at Jacobs University Bremen in Spring 2016.

Contents: The document mixes the slides presented in class with comments of the instructor to give students a more complete background reference.

Caveat: This document is made available for the students of this course only. It is still a draft and will develop over the course of the current course and in coming academic years.

Knowledge Representation Experiment: This document is also an experiment in knowledge representation. Under the hood, it uses the sTeX package [Koh08, Koh16], a TeX/LaTeX extension for semantic markup, which allows to export the contents into the eLearning platform PantaRhei.

Comments and extensions are always welcome, please send them to the author.

Other Resources: The course will be accompanied by a discussion forum on PantaRhei (`http://panta.kwarc.info/course-forum/1492`), which will be used for all announcements and can be used by the students for questions, discussions, and feedback. The course readings, course notes (this document), homework assignments, quizzes, and their solutions will be posted on `http://kwarc.info/teaching/TopModCS`.

Comments: Comments and extensions are always welcome, please send them to the author.

Acknowledgments

Sources: Parts of this course have been used in other lectures, and have been shaped by interdisciplinary discussions with my colleagues, in paricular Prof. Thomas Rommel and Dr. Giselda Beaudin.

TopModCS Students: The following students have submitted corrections and suggestions to this and earlier versions of the notes: Dennis Ledwon.

Recorded Syllabus for 2016

In this document, we record the progress of the course in spring 2016 in the form of a "recorded syllabus", i.e. a syllabus that is created after the fact rather than before.

Recorded Syllabus Spring Semester 2016:

#	date	until	slide	page
1	Feb. 2.	admin, digial documents	??	??
2	Feb. 9.	recap, basic data strutures	??	??
3	Feb. 16.	URIs	33	32
4	Feb. 23	HTML	45	39
5	Mar. 1.	Server Side Scripting	54	44
6.	Mar. 8.	XML	61	49
7.	Mar. 15.	XPath	68	53
8.	Mar. 29.	EPUB	76	58
9.	Apr. 5.	Revision Control /Merge	85	68
10.	Apr. 12.	Working with GIT	92	73
11.	Apr. 19.	Issue Traking Systems	102	78

Contents

Chapter 1

Administrativa

We will now go through the ground rules for the course. This is a kind of a social contract between the instructors and the students. Both have to keep their side of the deal to make the acquaintance with modern topics of computer science as efficient and painless as possible.

1.1 Resources

Even though the lecture itself will be the main source of information in the course, there are various resources from which to study the material.

Textbooks, Handouts and Information, Forum

- No required textbook, but course notes, posted slides
- Information resources (e.g. Course notes) will be posted at http://kwarc.info/teaching/TopModCS
- Everything will be posted on PantaRhei (Notes+assignments+course forum)
  - announcements, contact information, course schedule and calendar
  - discussion among your fellow students(careful, I will occasionally check for academic integrity!)
  - http://panta.kwarc.info (use your Jacobs login)
  - Set Up PantaRhei Access: to get notifications
    1) Log into http://panta.kwarc.info, (use your Jacobs login)
    2) find the course "TopModCS Spring 2016", (this course)
    3) request membership (I will approve you)
  - if there are problems contact the TAs or me.

No Textbook: Due to the special circumstances discussed above, there is no single textbook that covers the course. Instead we have a comprehensive set of course notes (this document). They are provided in two forms: as a large PDF that is posted at the course web page and on the PantaRhei system. The latter is actually the preferred method of interaction with the course materials, since it allows to discuss the material in place, to play with notations, to give feedback, etc. The PDF file is for printing and as a fallback, if the PantaRhei system, which is still under development, develops problems.

But of course, there is a wealth of literature on the subject, and the references at the end of the lecture notes can serve as a starting point for further reading. We will try to point out the relevant literature throughout the notes.

Software/Hardware tools

- ▷ You will need computer access for this course
- ▷ we recommend the use of standard software tools
  - ▷ find a text editor you are comfortable with (get good with it)
    A text editor is a program you can use to write text files. (not `MS Word`)
  - ▷ any operating system you like (I can only help with `UNIX`)
  - ▷ Any browser you like (I use `FireFox`: just a better browser (for Math))
- ▷ learn how to touch-type NOW (reap the benefits earlier, not later)

Touch-typing: You should not underestimate the amount of time you will spend typing during your studies. Even if you consider yourself fluent in two-finger typing, touch-typing will give you a factor two in speed. This ability will save you at least half an hour per day, once you master it. Which can make a crucial difference in your success.

Touch-typing is very easy to learn, if you practice about an hour a day for a week, you will re-gain your two-finger speed and from then on start saving time. There are various free typing tutors on the network. At `http://typingsoft.com/all_typing_tutors.htm` you can find about programs, most for windows, some for linux. I would probably try `Ktouch` or `TuxType`

Darko Pesikan (one of the previous TAs) recommends the `TypingMaster` program. You can download a demo version from `http://www.typingmaster.com/index.asp?go=tutordemo`

You can find more information by googling something like "`learn to touch-type`".

1.2 Grades, Homeworks, Submission, and Cheating

Now we come to a topic that is always interesting to the students: the grading scheme.

Prerequisites, Requirements, Grades

- ▷ Prerequisites: Motivation, Interest, Curiosity, hard work
  - ▷ in particular no prerequisites from Computer Science (self-contained)
  - ▷ knowing how to program helps understand, but is not necessary
  - ▷ you can do this course if you want! (and I want you to succeed)
- ▷ Grades:

Component	%	Comment
Homework	30	Weekly Assignments
Quizzes	30	first 10 minutes of class
Project	30	
Attendance and Wakefulness	10	I can watch you!

My main motivation in this grading scheme is to entice you to study continuously and keep up with the course. Therefore we have almost three-quarters of the grade dedicated to weekly components: *i*) the quizzes to make sure that you are well-prepared for class, *ii*) "A&W" to make sure you participate in class, and *iii*) the homeworks to give you a chance to play with the concepts presented in class and to understand them more thoroughly. For this I am willing to forego all exams; the only "global" grade component is a project where you can drill in on some particular part of the course contents and get your hands dirty.

Homework assignments

- Goal: Reinforce and apply what is taught/discussed in class.
- Homeworks: will be practical analysis/writing/programming assignments in a variety of formats (take time to solve)
- Admin: To keep things running smoothly
  - Homeworks will be posted on PantaRhei
  - Homeworks are handed in electronically in `JGrader`(plain text, Postscript, PDF,. . .)
  - discuss problems on PantaRhei (Profs/TAs/students can help you!)
- Homework discipline:
  - start early! (many assignments need more than one evening's work)
  - Don't start by sitting at a blank screen
  - Humans will be trying to understand the text/code when grading it.

Homework assignments are a central part of the course, they allow you to review the concepts covered in class, and practice using them.

Homework Submissions, Grading, Tutorials

- Submissions: We use Heinrich Stamerjohanns' `JGrader` system
  - submit all homework assignments electronically to `https://jgrader.de`.
  - you can login with your Jacobs account and password. (should have one!)
  - feedback/grades to your submissions
  - get an overview over how you are doing! (do not leave to midterm)

▷ Tutorials: select a tutorial group and actually go to it regularly

- ▷ to discuss the course topics after class (lectures need pre/postparation)
- ▷ to discuss your homework after submission (to see what was the problem)
- ▷ to find a study group (probably the most determining factor of success)

The next topic is very important, you should take this very seriously, even if you think that this is just a self-serving regulation made by the faculty.

All societies have their rules, written and unwritten ones, which serve as a social contract among its members, protect their interestes, and optimize the functioning of the society as a whole. This is also true for the community of scientists worldwide. This society is special, since it balances intense cooperation on joint issues with fierce competition. Most of the rules are largely unwritten; you are expected to follow them anyway. The code of academic integrity at Jacobs is an attempt to put some of the aspects into writing.

It is an essential part of your academic education that you learn to behave like academics, i.e. to function as a member of the academic community. Even if you do not want to become a scientist in the end, you should be aware that many of the people you are dealing with have gone through an academic education and expect that you (as a graduate of Jacobs) will behave by these rules.

The Code of Academic Integrity

▷ Jacobs has a "Code of Academic Integrity"

- ▷ this is a document passed by the Jacobs community (our law of the university)
- ▷ you have signed it during enrollment (we take this seriously)

▷ It mandates good behaviors from both faculty and students and penalizes bad ones:

- ▷ honest academic behavior (we don't cheat/falsify)
- ▷ respect and protect the intellectual property of others (no plagiarism)
- ▷ treat all Jacobs members equally (no favoritism)

▷ this is to protect you and build an atmosphere of mutual respect

- ▷ academic societies thrive on reputation and respect as primary currency

▷ The Reasonable Person Principle (one lubricant of academia)

- ▷ we treat each other as reasonable persons
- ▷ the other's requests and needs are reasonable until proven otherwise
- ▷ but if the other violates our trust, we are deeply disappointed (severe uncompromising consequences)

To understand the rules of academic societies it is central to realize that these communities are driven by economic considerations of their members. However, in academic societies, the primary good that is produced and consumed consists in ideas and knowledge, and the primary currency involved is academic reputation[1]. Even though academic societies may seem as altruistic — scientists share their knowledge freely, even investing time to help their peers understand the concepts more deeply — it is useful to realize that this behavior is just one half of an economic transaction. By publishing their ideas and results, scientists sell their goods for reputation. Of course, this can only work if ideas and facts are attributed to their original creators (who gain reputation by being cited). You will see that scientists can become quite fierce and downright nasty when confronted with behavior that does not respect other's intellectual property.

[1] Of course, this is a very simplistic attempt to explain academic societies, and there are many other factors at work there. For instance, it is possible to convert reputation into money: if you are a famous scientist, you may get a well-paying job at a good university,...

Chapter 2

Outline of the Course

Digital Documents in the Small and in the Large: In this course we will introduce and discuss the main concepts and technologies behind digital documents. We start out with a very brief overview over computing and programming as a basis – we do not cover them in this course, but an inkling of how they work is helpful to understand the concepts.

After this, we address how documents are encoded (stored in the computer and on disk and transmitted between computers), and then go into documents with markup (style information).

Finally, we address the issue of how to organize and interrelate large collections of (multimedia) documents: the world-wide-web.

In this course, we want to achieve two things: we want to

1) expose you to the concepts, structures, and problems in dealing with information and digital objects, in particular with digital documents
2) show you exemplarily the methods Computer Science uses to address such problems.

Outline: Dealing with Digital Documents

- ▷ Documents in the small and in the Large
  - ▷ Encoding Numbers and Characters
  - ▷ Documents & their meaning
  - ▷ Web & XML Technologies
- ▷ Mechanics of Documents: Formats & Tools
  - ▷ Legal Foundations
  - ▷ Computing with Documents
  - ▷ Electronic Books and Documents
  - ▷ Revision Control & Project Management
- ▷ Intelligent Documents/Media and the Future
  - ▷ Knowledge Representation & Semantic Web
  - ▷ Domain-Specific Languages/Formats: MathML & SVG
  - ▷ Active Documents

2.1 15 Minutes Introduction to Programming

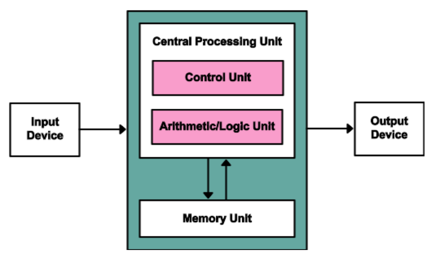
Programming is an important and distinctive part of "Computer Science" – the topic of this course. Even though we are not going to learn any programming in this course it is important to have some understanding of it. Therefore we go over the basics now.

To understand programming, it is important to realize that that computers are universal machines. Unlike a conventional tool – e.g a spade – which has a limited number of purposes/behaviors – digging holes in case of a spade, maybe hitting someone over the head, a computer can be given arbitrary[1] purposes/behaviors by specifying them in form of a "program".

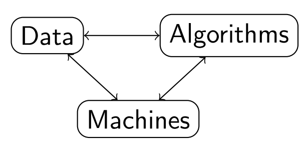
This notion of a program as a behavior specification for an universal machine is so powerful, that the field of computer science is centered around studying it – and what we can do with programs, this includes *i*) storing and manipulating data about the world, *ii*) encoding, generating, and interpreting images, audio, and video, *iii*) transporting information for communication, *iv*) representing knowledge and reasoning, *v*) transforming, optimizing, and verifying other programs, *vi*) learning patterns in data and predicting the future from the past.

Computer Hardware/Software & Programming

- ▷ **Definition 2.1.1** computer hardware consists of devices that execute commands/instructions:
  - ▷ central processing unit (CPU)
  - ▷ memory: e.g. RAM, Disks, ...
  - ▷ input: e.g. keyboard, touchscreen, ...
  - ▷ output: e.g. screen, earphone, ...



- ▷ software = data and programs
  - ▷ data represents the world
  - ▷ programs input, manipulate, output data
- ▷ hardwarehardware stores data and runs programs.



- ▷ Programming = writing programs (Telling the computer what to do)
- ▷ The computer does exactly as told
  - ▷ extremely fast extremely reliable
  - ▷ completely stupid: will not do what you mean unless you tell it exactly
- ▷ Programming can be extremely fun/frustrating/addictive (try it)

A universal machine has to have – so experience in computer science shows – certain distinctive parts.

- A CPU that consists of a
  - control unit that interprets the program and controls the flow of instructions and
  - a arithmetic/logic unit that does the actual computations internally.

[1] as long as they are "computable", not all are.

- Memory that allows the system to store data during runtime (volatile storage; usually RAM) and between runs of the system (persistant storage; usually hard disks, solid state disks, magnetic tapes, or optical media).
- I/O devices for the communication with the user and other computers.

With these components we can build various kinds of universal machines; these range from thought experiments like Turing machines, to today's general purpose computers like your laptop with various embedded computers (wristwatches, Internet routers, airbag controllers, ...) in-between.

Note that – given enough fantasy – the human brain has the same components Indeed the human mind is a universal machine – we can think whatever we want, react to the environment, and are not limited to particular behaviors. There is a sub-field of Computer Science that studies this: Artificial Intelligence (AI). In this analogy, the brain is the "hardware" –sometimes called "wetware" because it is not made of hard silicon or "meat machine"[2]. It is instructional to think about what the program and the data might be in this analogy.

AI studies human intelligence with the premise that the brain is a computational machine and that intelligence is a "program" running on it. In particular, the working hypothesis is that we can "program" intelligence. Even though AI has many successful applications, it has not succeeded in creating a machine that exhibits the equivalent to general human intelligence, so the jury is still out whether the AI hypothesis is true or not. In any case it is a fascinating area of scientific inquiry.

Note: this has an immediate consequence for the discussion in our course. Even though computers can execute programs very efficiently, you should not expect them to "think" like a human. In particular, they will execute programs exactly as you have written them. This has two consequences:

- the behavior of programs is – in principle – predictable
- all errors of program behavior are your own (the programmer's)

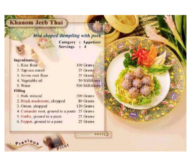
We can now have a closer look at program execution and programs. Before we go into the detalis of how programs look concretely, let us fix some concepts.

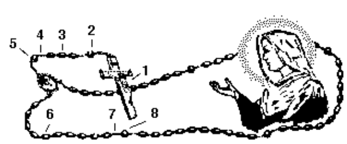
In computer science, we distinguish two levels on which we can talk about programs. The more general is the level of algorithms, which is independent of the concrete programming language. Algorithms express the general ideas and flow of computation and can be realized in various languages, but are all equivalent – in terms of the algorithm they implement.

As they are not bound to programming languages algorithms transcend them, and we can find them in our daily lives, e.g. as sequences of instructions like recipes, grame instructions, and the like. This should make algorithms quite familiar; the only difference of programs is that they are written down in an unambiguous syntax that a computer can understand.

Program Execution

▷ Algorithm: informal description of what to do (good enough for humans)





▷ Program: computer-processable version, e.g. in python

```
for x in range(0, 3):
    print ("we tell you",x,"time(s)")
```

▷ Interpreter: reads a program and executes it directly

[2]Marvin Minsky; one of the founders of AI

- ▷ special case: interactive interpretation (lets you experiment easily)

▷ Compiler: translates a program (the source) into another program (the binary) in a much simpler language for optimized execution on hardware directly.

▷ **Remark 2.1.2** Compilers are efficient, but more cumbersome for development.

We have two kinds of programming languages: one which the CPU can execute directly – these are very very difficult for humans to understand and maintain – and higher-level ones that are understandable by humans. If we want to use high-level languages – and we do, then we need to have some way bridging the language gap: this is what compilers and interpreters do.

Programming Languages

▷ The language in which we write the program

- ▷ formal, symbolic, precise meaning

▷ There are lots of programming languages

- ▷ design huge effort in computer science
- ▷ all programming languages equally strong
- ▷ each is more or less appropriate for a specific task depending on the circumstances

▷ Lots of paradigms: imperative, functional programming, logic programming, object oriented programming

▷ Everybody who tells you that one PL is the best has no idea what they're talking about

This concludes our ultra-brief recap of programming. Of course it is much better to get some first-hand exposure to programming; at Jacobs university we have the python courses for this, we highly recommend them: If you really want to understand programming you have to get your hands dirty and do it.

Part I

Plain Text Files

Chapter 3

Documents as Digital Objects

In this Chapter we take a first look at documents and how they are represented on the computer.

Documents as Digital Objects

- Question: how do texts get onto the computer?(after all, computers can only do 0/1)
- Hint: At the most basic level, texts are just sequences of characters.
- Answer: We have to encode characters as sequences of bits.
- We will go into how:
  - documents are represented as sequences of characters
  - characters are represented as numbers
  - numbers are represented as bits (0/1)

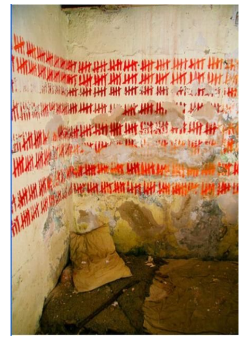
3.1 Representing and Manipulating Numbers

We start with the representation of numbers. There are multiple number systems, as we are interested in the principles only, we restrict ourselves to the natural numbers – all other number systems can be built on top of these. But even there we have choices about representation, which influence the space we need and how we compute with natural numbers.

The first system for number representations is very simple; so simple in fact that it has been discovered and use a long time ago.

Natural Numbers

- Numbers are symbolic representations of numeric quantities.
- There are many ways to represent numbers (more on this later)
- let's take the simplest one (about 8,000 to 10,000 years old)

- ▷ we count by making marks on some surface.
- ▷ For instance //// stands for the number four (be it in 4 apples, or 4 worms)

In addition to manipulating normal objects directly linked to their daily survival, humans also invented the manipulation of place-holders or symbols. A *symbol* represents an object or a set of objects in an abstract way. The earliest examples for symbols are the cave paintings showing iconic silhouettes of animals like the famous ones of Cro-Magnon. The invention of symbols is not only an artistic, pleasurable "waste of time" for mankind, but it had tremendous consequences. There is archaeological evidence that in ancient times, namely at least some 8000 to 10000 years ago, men started to use tally bones for counting. This means that the symbol "bone with marks" was used to represent numbers. The important aspect is that this bone is a symbol that is completely detached from its original down to earth meaning, most likely of being a tool or a waste product from a meal. Instead it stands for a universal concept that can be applied to arbitrary objects.

So far so good, let us see how this would be represented on a computer:

Unary Natural Numbers on the Computer

- ▷ **Definition 3.1.1** We call the representation of natural numbers by slashes on a surface the unary natural numbers
- ▷ Question: How do we represent them on a computer? (not bones or walls)
- ▷ Idea: If we have a memory bank of n binary digits, initialize all by 0, represent each slash by a 1 from the right.
- ▷ **Example 3.1.2** Memory bank with 32 binary digits, represening 11.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1

Problem: For realistic arithmetics we need better number representations than the unary natural numbers (e.g. for representing the number of EU citizens $\hat{=}$ 100 000 pages of /)

The unary natural numbers are very simple and direct, but they are neither space-efficient, nor easy to manipulate. Therefore we will use different ways of representing numbers in practice.

▷ Positional Number Systems

▷ Problem: Find a better representation system for natural numbers.

▷ Idea: build a clever code on the unary numbers, use position information and addition, multiplication, and exponentiation.

▷ **Definition 3.1.3** A positional number system $\mathcal{N}$ is a pair $\mathcal{N} = \langle D_b, \varphi_b \rangle$ with

- ▹ D_b is a finite alphabet of b digits. b is called the base or radix of $\mathcal{N}$
- ▹ assign each digit $d \in D_b$ a number $\varphi_b(d)$ between 0 and $b-1$.
- ▹ Extend φ_b to sequences of digits by $\varphi_b(\langle n_k, \ldots, n_1 \rangle) := \sum_{i=1}^{k} \varphi_b(n_i) \cdot b^{i-1}$

▷ **Example 3.1.4** $\langle \{a, b, c\}, \varphi \rangle$ with with $\varphi(a) := 0$, $\varphi(b) := 1$, and $\varphi(c) := 2$ is a positional number system for base three. We have

$$\varphi(\langle c, a, b \rangle) = 2 \cdot 3^2 + 0 \cdot 3^1 + 1 \cdot 3^0 = 18 + 0 + 1 = 19$$

▷ **Observation 3.1.5** *To convert a number n to base b, use successive integer division (division with remainder) by b:*

$i := n$; repeat (record $i \bmod b$, $i := i \operatorname{div} b$) until $i = 0$.

▷ **Example 3.1.6 (Convert 456 to base 8)** Result: 710_8

$$\begin{array}{ll} 456 \operatorname{div} 8 = 57 & 456 \bmod 8 = 0 \\ 57 \operatorname{div} 8 = 7 & 57 \bmod 8 = 1 \\ 7 \operatorname{div} 8 = 0 & 7 \bmod 8 = 7 \end{array}$$

The problem with the unary number system is that it uses enormous amounts of space, when writing down large numbers. We obviously need a better encoding.

If we look at the unary number system from a greater distance, we see that we are not using a very important feature of strings here: position. As we only have one letter in our alphabet (/), we cannot, so we should use a larger alphabet. The main idea behind a positional number system $\mathcal{N} = \langle D_b, \varphi_b \rangle$ is that we encode numbers as strings of digits in D_b, such that the position matters, and to give these encoding a meaning by mapping them into the unary natural numbers via a mapping φ_b. This is the the same process we did for the logics; we are now doing it for number systems. However, here, we also want to ensure that the meaning mapping φ_b is a bijection, since we want to define the arithmetics on the encodings by reference to The arithmetical operators on the unary natural numbers.

Commonly Used Positional Number Systems

▷ **Example 3.1.7** The following positional number systems are in common use.

name	set	base	digits	example
unary	$\mathbb{N}_1$	1	/	$/////_1$
binary	$\mathbb{N}_2$	2	0,1	0101000111_2
octal	$\mathbb{N}_8$	8	0,1,...,7	63027_8
decimal	$\mathbb{N}_{10}$	10	0,1,...,9	162098_{10} or 162098
hexadecimal	$\mathbb{N}_{16}$	16	0,1,...,9,A,...,F	$FF3A12_{16}$

▷ **Notation 3.1.8** attach the base of $\mathcal{N}$ to every number from $\mathcal{N}$. (default: decimal)

Trick: Group triples or quadruples of binary digits into recognizable chunks(add leading zeros as needed)

▷ ▷ $110001101011100_2 = \underbrace{0110_2}_{6_{16}}\underbrace{0011_2}_{3_{16}}\underbrace{0101_2}_{5_{16}}\underbrace{1100_2}_{C_{16}} = 635C_{16}$

▷ $110001101011100_2 = \underbrace{110_2}_{6_8}\underbrace{001_2}_{1_8}\underbrace{101_2}_{5_8}\underbrace{011_2}_{3_8}\underbrace{100_2}_{4_8} = 61534_8$

▷ $F3A_{16} = \underbrace{F_{16}}_{1111_2}\underbrace{3_{16}}_{0011_2}\underbrace{A_{16}}_{1010_2} = 111100111010_2$, $4721_8 = \underbrace{4_8}_{100_2}\underbrace{7_8}_{111_2}\underbrace{2_8}_{010_2}\underbrace{1_8}_{001_2} = 100111010001_2$

We have all seen positional number systems: our decimal system is one (for the base 10). Other systems that important for us are the binary system (it is the smallest non-degenerate one) and the octal- (base 8) and hexadecimal- (base 16) systems. These come from the fact that binary numbers are very hard for humans to scan. Therefore it became customary to group three or four digits together and introduce we (compound) digits for them. The octal system is mostly relevant for historic reasons, the hexadecimal system is in widespread use as syntactic sugar for binary numbers, which form the basis for circuits, since binary digits can be represented physically by current/no current.

Arithmetics in Positional Number Systems

▷ For arithmetics just follow elementery school rules (for the right base)

▷ Tom Lehrer's "New Math"

▷ **Example 3.1.9**

Addition base 4

	1	2	3
+	1_1	2_1	3
	3	1	2

binary multiplication

		1	0	1	0
	*		1	1	0
		0	0	0	0
	1	0	1	0	
1	0	1	0		
1	1	1	1	0	0

3.2 Encoding Characters as Numbers

IT systems need to encode characters from our alphabets as bit strings (sequences of binary digits (bits) 0 and 1) for representation in computers. To understand the current state – the unicode standard – we will take a historical perspective.

It is important to understand that encoding and decoding of characters is an activity that requires standardization in multi-device settings – be it sending a file to the printer or sending an e-mail to a friend on another continent. Concretely, the recipient wants to use the same character mapping for decoding the sequence of bits as the sender used for encoding them – otherwise the message is garbled.

We observe that we cannot just specify the encoding table in the transmitted document itself, (that information would have to be en/decoded with the other content), so we need to rely document-external external methods like standardization or encoding negotiation at the meta-level. In this Section we will focus on the former.

The `ASCII` code we will introduce here is one of the first standardized and widely used character encodings for a complete alphabet. It is still widely used today. The code tries to strike a balance between a being able to encode a large set of characters and the representational capabilities in the time of punch cards (see below).

The ASCII Character Code

▷ **Definition 3.2.1** The American Standard Code for Information Interchange (`ASCII`) code assigns characters to numbers 0-127

Code	···0	···1	···2	···3	···4	···5	···6	···7	···8	···9	···A	···B	···C	···D	···E	···F
0···	teNUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1···	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2···		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3···	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4···	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5···	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6···	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7···	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

The first 32 characters are control characters for ASCII devices like printers

⊳ Motivated by punchcards: The character 0 (binary 0000000) carries no information `teNUL`, (used as dividers)
Character 127 (binary 1111111) can be used for deleting (overwriting) last value (cannot delete holes)

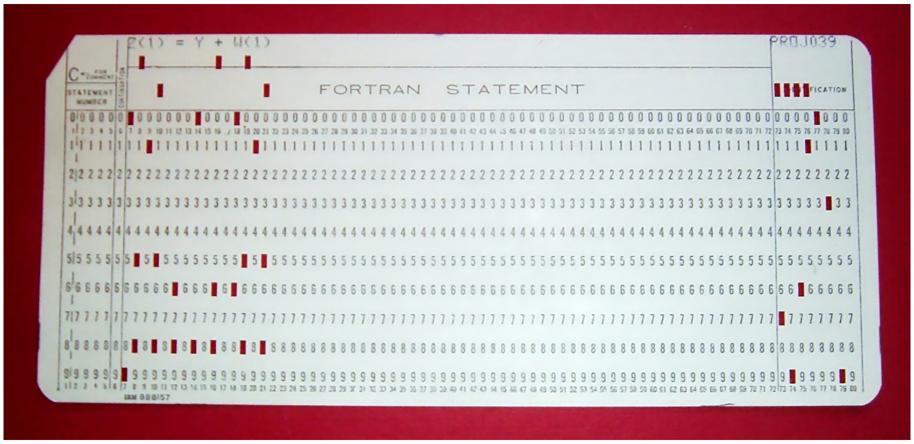
▷ The ASCII code was standardized in 1963 and is still prevalent in computers today (but seen as US-centric)

Punch cards were the the preferred medium for long-term storage of programs up to the late 1970s, since they could directly be produced by card punchers and automatically read by computers.

A Punchcard

▷ A punch card is a piece of stiff paper that contains digital information represented by the presence or absence of holes in predefined positions.

▷ **Example 3.2.2** This punch card encoded the `FORTRAN` statement `Z(1) = Y + W(1)`



Up to the 1970s, computers were batch machines, where the programmer delivered the program to the operator (a person behind a counter who fed the programs to the computer) and collected the printouts the next morning. Essentially, each punch card represented a single line (80 characters) of program code. Direct interaction with a computer is a relatively young mode of operation.

The `ASCII` code as above has a variety of problems, for instance that the control characters are mostly no longer in use, the code is lacking many characters of languages other than the English language it was developed for, and finally, it only uses seven bits, where a byte (eight bits) is the preferred unit in information technology. Therefore there have been a whole zoo of extensions, which — due to the fact that there were so many of them — never quite solved the encoding problem.

Problems with `ASCII` encoding

- Problem: Many of the control characters are obsolete by now (e.g. `teNUL`,`BEL`, or `DEL`)
- Problem: Many European characters are not represented (e.g. è,ñ,ü,ß,. . .)
- European ASCII Variants: Exchange less-used characters for national ones
- **Example 3.2.3 (German `ASCII`)** remap e.g. `[` ↦ `Ä`, `]` ↦ `Ü` in German `ASCII` (“`Apple ][`” comes out as “`Apple ÜÄ`”)
- **Definition 3.2.4 (ISO-Latin (ISO/IEC 8859))** 16 Extensions of ASCII to 8-bit (256 characters) ISO-Latin 1 ≙ “Western European”, ISO-Latin 6 ≙ “Arabic”,ISO-Latin 7 ≙ “Greek”. . .
- Problem: No cursive Arabic, Asian, African, Old Icelandic Runes, Math,. . .
- Idea: Do something totally different to include all the world’s scripts: For a scalable architecture, separate
  - what characters are available from the (character set)
  - bit string-to-character mapping (character encoding)

The goal of the UniCode standard is to cover all the worlds scripts (past, present, and future) and provide efficient encodings for them. The only scripts in regular use that are currently excluded are fictional scripts like the elvish scripts from the Lord of the Rings or Klingon scripts from the Star Trek series.

An important idea behind UniCode is to separate concerns between standardizing the character set — i.e. the set of encodable characters and the encoding itself.

Unicode and the Universal Character Set

▷ **Definition 3.2.5 (Twin Standards)** A scalable Architecture for representing all the worlds scripts

- ▷ The universal character set defined by the ISO/IEC 10646 International Standard, is a standard set of characters upon which many character encodings are based.
- ▷ The unicode Standard defines a set of standard character encodings, rules for normalization, decomposition, collation, rendering and bidirectional display order

▷ **Definition 3.2.6** Each UCS character is identified by an unambiguous name and an integer number called its code point.

▷ The UCS has 1.1 million code points and nearly 100 000 characters.

▷ **Definition 3.2.7** Most (non-Chinese) characters have code points in $[1, 65536]$ (the basic multilingual plane).

▷ **Notation 3.2.8** For code points in the Basic Multilingual Plane (BMP), four digits are used, e.g. $U+0058$ for the character $LATINCAPITALLETTERX$;

Note that there is indeed an issue with space-efficient encoding here. UniCode reserves space for 2^{32} (more than a million) characters to be able to handle future scripts. But just simply using 32 bits for every UniCode character would be extremely wasteful: UniCode-encoded versions of `ASCII` files would be four times as large.

Therefore UniCode allows multiple encodings. UTF-32 is a simple 32-bit code that directly uses the code points in binary form. UTF-8 is optimized for western languages and coincides with the `ASCII` where they overlap. As a consequence, `ASCII` encoded texts can be decoded in UTF-8 without changes — but in the UTF-8 encoding, we can also address all other UniCode characters (using multi-byte characters).

Character Encodings in Unicode

▷ **Definition 3.2.9** A character encoding is a mapping from bit strings to UCS code points.

▷ Idea: Unicode supports multiple encodings (but not character sets) for efficiency

▷ **Definition 3.2.10 (Unicode Transformation Format)**

- ▷ UTF-8, 8-bit, variable-width encoding, which maximizes compatibility with ASCII.
- ▷ UTF-16, 16-bit, variable-width encoding (popular in Asia)
- ▷ UTF-32, a 32-bit, fixed-width encoding (for safety)

▷ **Definition 3.2.11** The UTF-8 encoding follows the following encoding scheme

Unicode	Byte1	Byte2	Byte3	Byte4
U+ 000000 − U+ 00007F	0xxxxxxx			
U+ 000080 − U+ 0007FF	110xxxxx	10xxxxxx		
U+ 000800 − U+ 00$FFFF$	1110xxxx	10xxxxxx	10xxxxxx	
U+ 010000 − U+ 10$FFFF$	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

▷ **Example 3.2.12** \$ = U+ 0024 is encoded as 00100100 (1 byte)
¢ = U+ 00A2 is encoded as 11000010,10100010 (two bytes)
€ = U+ 20AC is encoded as 11100010,10000010,10101100 (three bytes)

Note how the fixed bit prefixes in the encoding are engineered to determine which of the four cases apply, so that UTF-8 encoded documents can be safely decoded..

3.3 Representing & Manipulating Documents on a Computer

Now that we can represent characters as bit sequences, we can represent text documents. In principle text documents are just sequences of characters; they can be represented by just concatenating them.

Digital Text

▷ **Definition 3.3.1** Digital text is a digital encoding of textual material that can be read without much processing.

▷ **Definition 3.3.2** Digital text is subdivided into plain text, where all characters carry the textual information and formatted text, which also contains markup codes in form of control words (character sequences) that specify formatting, meaning, or metadata. All characters that are not control words are constitute the textual content of formatted text.

▷ Even though formatted text can read directly, it is usually consumed by humans through a document viewer, i.e. a device that interprets the control words and visualizes the textual content accordingly.

▷ **Definition 3.3.3** A markup language is a specific system for markup codes for digital text.

File Types

▷ **Definition 3.3.4** A text file is a computer file that is structured as a sequence of lines of digital text. Computer files that are not text files are called binary files.

▷ **Remark 3.3.5** Text files are usually encoded with `ASCII`, ISO-Latin, or – increasingly – UniCode encodings like UTF-8.

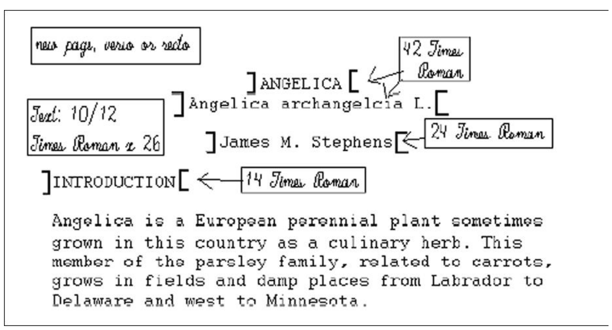
Remark 3.3.6 Plain text is different from formatted text, where style information is included and binary files in which some portions must be interpreted as binary objects (encoded integers, real numbers, images, etc.)

Document Markup

▷ **Definition 3.3.7** Document markup is the process of adding markup codes to a document to control the structure, formatting, or the relationship among its parts.

▷ **Remark 3.3.8** Document markup turns plain text into formatted text.

▷ **Example 3.3.9** A text with markup codes (for printing)



There are many systems for document markup ranging from informal ones as in ?document-markup.ex? that specify the intended document appearance to humans – in this case the printer – to technical ones which can be understood by machines but serving the same purpose.

Text Editors

▷ **Definition 3.3.10** A text editor is a program used for editing text files.

▷ **Example 3.3.11** Popular text editors include

- ▷ `Notepad` is a simple editor distributed with `Windows`.
- ▷ `emacs` and `vi` are powerful editors originating from `UNIX` and optimized for programming.

- sublime is a sophisticated programming editor for multiple operating systems.
- EtherPad is a browser-based real-time collaborative editor.

▷ **Example 3.3.12** Even though it can save documents as text files, MS Word is not usually considered a text editor, since it is optimized towards formatted text; such "editors" are called word processors.

 25 JACOBS UNIVERSITY

Word Processors and Formatted Text

▷ **Definition 3.3.13** A word processor is a software application, that performs the task of composition, editing, formatting, printing of documents represented as formatted text. The particular representation format is called the document format.

▷ **Example 3.3.14** Popular word processors include

- MS Word is an elaborated word processor for Windows, whose native format is Office Open XML (file extension .docx).
- OpenOffice and LibreOffice are similar word processors using the ODF format (Open Office Format; file extension .odf) natively, but can also import other formats..
- Pages is a word processors for Mac OS X it uses a proprietary format.
- Office Online and GoogleDocs are browser-based real-time collaborative word processors.

▷ **Example 3.3.15** Text editors are usually not considered to be word processors, even though they can sometimes be used to edit markup-based formatted text.

 26 JACOBS UNIVERSITY

3.4 Measuring Sizes of Documents/Units of Information

Having represented documents are sequenes of characters, we can use that to measure the sizes of documents. In this Section we will have a look at the underlying units of information and try to get an intuition about what we can store in files.

⚠2: We will take a very generous stance towards what a document is, in particular, we will include pictures, audio files, spreadsheets, computer aided designs, ...

Unis for Information

▷ Observation: The smallest unit of information is knowing the state of a system with only two states.

▷ **Definition 3.4.1** A bit (a contraction of "binary digit") is the basic unit of capacity of a data storage device or communication communication channel.

The capacity of a system which can exist in only two states, is one bit (written as $1\,\mathrm{b}$)

▷ Note: In the ASCII encoding, one character is encoded as $8\,\mathrm{b}$, so we introduce another basic unit:

▷ **Definition 3.4.2** The byte is a derived unit for information capacity: $1\,\mathrm{B} = 8\,\mathrm{b}$.

From the basic units of information, we can make prefixed units for prefixed units for larger chunks of information. But note that the usual SI unit prefixes are inconvenient for application to information measures, since powers of two are much more natural to realize (recall the discussion on balanced binary trees).

Larger Units of Information via Binary Prefixes

▷ We will see that memory comes naturally in powers to 2, as we address memory cells by binary numbers, therefore the derived information units are prefixed by special prefixes that are based on powers of 2.

▷ **Definition 3.4.3 (Binary Prefixes)** The following binary unit prefix es are used for information units because they are similar to the SI unit prefixes.

prefix	symbol	2^n	decimal	~SI prefix	Symbol
kibi	Ki	2^{10}	1024	kilo	k
mebi	Mi	2^{20}	1048576	mega	M
gibi	Gi	2^{30}	1.074×10^9	giga	G
tebi	Ti	2^{40}	1.1×10^{12}	tera	T
pebi	Pi	2^{50}	1.125×10^{15}	peta	P
exbi	Ei	2^{60}	1.153×10^{18}	exa	E
zebi	Zi	2^{70}	1.181×10^{21}	zetta	Z
yobi	Yi	2^{80}	1.209×10^{24}	yotta	Y

Note: The correspondence works better on the smaller prefixes; for yobi vs. yotta there is a 20% difference in magnitude.

⊳▷ The SI unit prefixes (and their operators) are often used instead of the correct binary ones defined here.

▷ **Example 3.4.4** You can buy hard-disks that say that their capacity is "one tera-byte", but they actually have a capacity of one tebibyte.

Let us now look at some information quantities and their real-world counterparts to get an intuition for the information content.

How much Information?

Bit (b)	*binary digit 0/1*
Byte (B)	*8 bit*
2 Bytes	A Unicode character in UTF.
10 Bytes	your name.
Kilobyte (**k** B)	*1,000 bytes OR* 10^3 *bytes*
2 Kilobytes	A Typewritten page.
100 Kilobytes	A low-resolution photograph.
Megabyte (**M** B)	*1,000,000 bytes OR* 10^6 *bytes*
1 Megabyte	A small novel or a 3.5 inch floppy disk.
2 Megabytes	A high-resolution photograph.
5 Megabytes	The complete works of Shakespeare.
10 Megabytes	A minute of high-fidelity sound.
100 Megabytes	1 meter of shelved books.
500 Megabytes	A CD-ROM.
Gigabyte (**G** B)	*1,000,000,000 bytes or* 10^9 *bytes*
1 Gigabyte	a pickup truck filled with books.
20 Gigabytes	A good collection of the works of Beethoven.
100 Gigabytes	A library floor of academic journals.

How much Information?

Terabyte (**T** B)	*1,000,000,000,000 bytes or* 10^{12} *bytes*
1 Terabyte	50000 trees made into paper and printed.
2 Terabytes	An academic research library.
10 Terabytes	The print collections of the U.S. Library of Congress.
400 Terabytes	National Climate Data Center (NOAA) database.
Petabyte (**P** B)	*1,000,000,000,000,000 bytes or* 10^{15} *bytes*
1 Petabyte	3 years of EOS data (2001).
2 Petabytes	All U.S. academic research libraries.
20 Petabytes	Production of hard-disk drives in 1995.
200 Petabytes	All printed material (ever).
Exabyte (**E** B)	*1,000,000,000,000,000,000 bytes or* 10^{18} *bytes*
2 Exabytes	Total volume of information generated in 1999.
5 Exabytes	All words ever spoken by human beings ever.
300 Exabytes	All data stored digitally in 2007.
Zettabyte (**Z** B)	*1,000,000,000,000,000,000,000 bytes or* 10^{21} *bytes*
2 Zettabytes	Total volume digital data transmitted in 2011
100 Zettabytes	Data equivalent to the human Genome in one body.

The information in this table is compiled from various studies, most recently [HL11].

Note: Information content of real-world artifacts can be assessed differently, depending on the view. Consider for instance a text typewritten on a single page. According to our definition, this has ca. 2k B, but if we fax it, the image of the page has 2M B or more, and a recording of a text read out loud is ca. 50M B. Whether this is a terrible waste of bandwidth depends on the application. On a fax, we can use the shape of the signature for identification (here we actually care more about the shape of the ink mark than the letters it encodes) or can see the shape of a coffee stain. In the audio recording we can hear the inflections and sentence melodies to gain an impression on the emotions that come with text.

Part II

Web and XML Technologies for Documents

Chapter 4

Basic Concepts of the World Wide Web

4.1 Preliminaries

The World Wide Web (WWWeb) is the hypertext/multimedia part of the Internet. It is implemented as a service on top of the Internet (at the application level) based on specific protocols and markup formats for documents.

The Internet and the Web

▷ **Definition 4.1.1** The Internet is a worldwide computer network that connects hundreds of thousands of smaller networks. (The mother of all networks)

▷ **Definition 4.1.2** The World Wide Web (WWW or WWWeb) is an open source information space where documents and other web resources are identified by URLs, interlinked by hypertext links, and can be accessed via the Internet.

▷ The WWW is the multimedia part of the Internet, they form critical infrastructure for modern society and commerce.

▷ The Internet/WWW is huge:

Year	Web	Deep Web	eMail
1999	21 TB	100 TB	11TB
2003	167 TB	92 PB	447 PB
2010	????	?????	?????

▷ We want to understand how it works (services and scalability issues)

.

 31 JACOBS UNIVERSITY

Given this recap we can now introduce some vocabulary to help us discuss the phenomena.

Concepts of the World Wide Web

- **Definition 4.1.3** A web page is a document on the WWWeb that can include multimedia data and hyperlinks.
- **Definition 4.1.4** A web site is a collection of related Web pages usually designed or controlled by the same individual or company.
- a web site generally shares a common domain name.
- **Definition 4.1.5** A hyperlink is a reference to data that can immediately be followed by the user or that is followed automatically by a user agent.
- **Definition 4.1.6** A collection text documents with hyperlinks that point to text fragments within the collection is called a hypertext. The action of following hyperlinks in a hypertext is called browsing or navigating the hypertext.
- In this sense, the WWWeb is a multimedia hypertext.

4.2 Addressing on the World Wide Web

The essential idea is that the World Wide Web consists of a set of resources (documents, images, movies, etc.) that are connected by links (like a spider-web). In the WWWeb, the the links consist of pointers to addresses of resources. To realize them, we only need addresses of resources (much as we have IP numbers as addresses to hosts on the Internet).

Uniform Resource Identifier (URI), Plumbing of the Web

- **Definition 4.2.1** A uniform resource identifier (URI) is a global identifiers of network-retrievable documents (web resources). URIs adhere a uniform syntax (grammar) defined in RFC-3986 [BLFM05]. A URI is made up of the following components:
  - a scheme that specifies the protocol governing the resource
  - an authority: the host (authentification there) that provides the resource.
  - a path in the hierarchically organized resources on the host.
  - a query in the non-hierarchically organized part of the host data.
  - a fragment identifer in the resource.
- **Example 4.2.2** The following are two example URIs and their component parts:

```
  http://example.com:8042/over/there?name=ferret#nose
  \__/   \______________/\_________/ \_________/ \__/
   |           |            |            |         |
scheme     authority       path        query   fragment
   |___   __________________|__________
  /    \ /                             \
 mailto:m.kohlhase@jacobs-university.de
```

Note: URIs only identify documents, they do not have to be provide access to them (e.g. in a browser).

The definition above only specifies the structure of a URI and its functional parts. It is designed to cover and unify a lot of existing addressing schemes, including URLs (which we cover next), ISBN numbers (book identifiers), and mail addresses.

In many situations URIs still have to be entered by hand, so they can become quite unwieldy. Therefore there is a way to abbreviate them.

▷ Relative URIs

▷ **Definition 4.2.3** uri-nutshellsURI can be abbreviated to relative URIs; missing parts are filled in from the context.

▷ **Example 4.2.4** Relative URIs are more convenient to write

relative URI	abbreviates	in context
`#foo`	⟪`current-file`⟫`#foo`	curent file
`bar.txt`	`file:///home/kohlhase/foo/bar.txt`	file system
`../bar/bar.html`	`http://example.org/bar/bar.html`	on the web

▷ **Definition 4.2.5** To distinguish them from relativesURI, we call URIs absolute URIs.

The important concept to grasp for relative URIs is that the missing parts can be reconstructed from the context they are found in: the document itself and how it was retrieved.

For the file system example, we are assuming that the document is a file `foo.html` that was loaded from the file system – under the file system URI `file:///home/kohlhase/foo/foo.html` – and for the web example via the URI `//example.org/foo/foo.html`. Note that in the last example, the relative URI `../bar/` goes up one segment of the path component (that is the meaning of `../`), and specifies the file `bar.html` in the directory `bar`.

But relative URIs have another advantage over absolute URIs: they make a web page or web site easier to move. If a web site only has links using relative URIs internally, then those do not mention e.g. authority (this is recovered from context and therefore variable), so we can freely move the web-site e.g. between domains.

Note that some forms of URIs can be used for actually locating (or accessing) the identified resources, e.g. for retrieval, if the resource is a document or sending to, if the resource is a mailbox. Such URIs are called "uniform resource *locators*", all others "uniform resource *locators*".

Uniform Resource Names and Locators

▷ **Definition 4.2.6** A uniform resource locator (URL) is a URI that that gives access to a web resource, by specifying an access method or location. All other URIs are called uniform resource names (URN).

▷ Idea: A URN defines the identity of a resource, a URL provides a method for finding it.

▷ **Example 4.2.7** The following URI is a URL (try it in your browser)
`http://kwarc.info/kohlhase/index.html`

▷ **Example 4.2.8** `urn:isbn:978-3-540-37897-6` only identifies [Koh06] (it is in the library)

▷ **Example 4.2.9** URNs can be turned into URL via a catalog service, e.g. `http://wm-urn.org/urn:isbn:978-3-540-37897-6`

▷ Note: URI/URLs are one of the core features of the web infrastructure, they are considered to be the plumbing of the WWWeb. (direct the flow of data)

Historically, started out as URLs as short strings used for locating documents on the Internet. The generalization to identifiers (and the addition of URNs) as a concept only came about when the concepts evolved and the application layer of the Internet grew and needed more structure.

Note that there are two ways in URIs can fail to be resource locators: first, the scheme does not support direct access (as the ISBN scheme in our example), or the scheme specifies an access method, but address does not point to an actual resource that could be accessed. Of course, the problem of "dangling links" occurs everywhere we have addressing (and change), and so we will neglect it from our discussion. In practice, the URL/URN distinction is mainly driven by the scheme part of a URI, which specifies the access/identification scheme.

Internationalized Resource Identifiers

▷ **Remark 4.2.10** URIs are `ASCII` strings.

▷ Problem: This is awkward e.g. for France Télécom, worse in Asia.

▷ Solution?: Use unicode (no, too young/unsafe)

▷ **Definition 4.2.11** Internationalized resource identifiers (IRIs) extend the `ASCII`-based URIs to the universal character set.

▷ **Definition 4.2.12** The URI encoding maps non-`ASCII` character to a `ASCII` string:

1) map character to its UTF-8 representation
2) represent each byte of the UTF-8 representation by three characters.
3) The first character is the percent sign (%),
4) and the other two characters are the hexadecimal representation of the byte.

▷ **Example 4.2.13** The letter "ł" (U+ 142) would be represented as %C5%82.

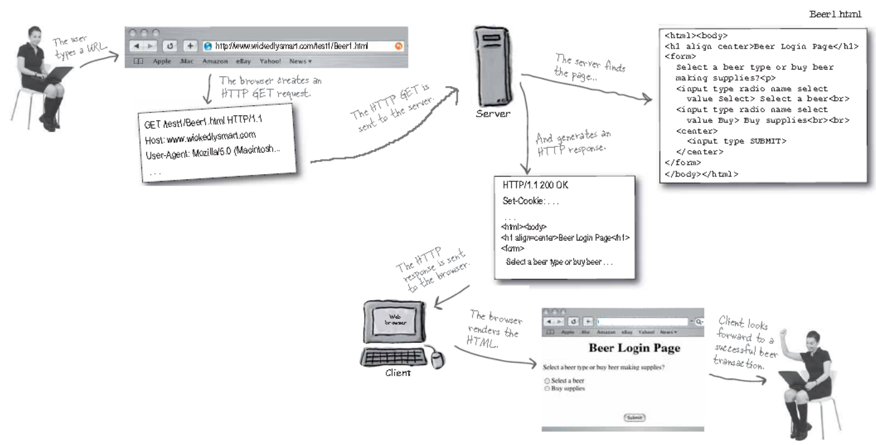
▷ **Example 4.2.14** `http://www.Übergrößen.de` becomes `http://www.%C3%9Cbergr%C3%B6%C3%9Fen.de`

▷ **Remark 4.2.15** Your browser can still show the URI-decoded version (so you can read it)

4.3 Running the World Wide Web

The infrastructure of the WWWeb relies on a client-server architecture, where the servers (called web servers) provide documents and the clients (usually web browsers) present the documents to the (human) users. Clients and servers communicate via the http protocol. We give an overview via a concrete example before we go into details.

The World Wide Web as a Client/Server System



 37

We will now go through and introduce the infrastructure components of the WWWeb in the order we encounter them. We start with the user agent; in our example the web browser used by the user to request the web page by entering its URL into the URL bar.

Web Browsers

▷ **Definition 4.3.1** A web browser is a software application for retrieving, presenting, and traversing information resources on the World Wide Web, enabling users to view web pages and to jump from one page to another.

▷ Practical Browser Tools:

- ▷ Status Bar: security info, page load progress
- ▷ Favorites (bookmarks)
- ▷ View Source: view the code of a Web page
- ▷ Tools/Internet Options, history, temporary Internet files, home page, auto complete, security settings, programs, etc.

▷ **Example 4.3.2 (Common Browsers)** ▷ `MS Internet Explorer` is provided by Microsoft for `Windows` (very common)

- ▷ `FireFox` is an open source browser for all platforms, it is known for its standards compliance.
- ▷ `Safari` is provided by Apple for `Mac OS X` and `Windows`

- Chrome is a lean and mean browser provided by Google
- WebKit is a library that forms the open source basis for Safari and Chrome.

The web browser communicates with the web server through a specialized protocol, the hypertext transfer protocol, which we cover now.

HTTP: Hypertext Transfer Protocol

- **Definition 4.3.3** The Hypertext Transfer Protocol (HTTP) is an application layer protocol for distributed, collaborative, hypermedia information systems.
- June 1999: HTTP/1.1 is defined in RFC 2616 [FGM+99].

  Definition 4.3.4 HTTP is used by a client (called user agent) to access web resources (addressed by Uniform Resource Locators (URLs)) via a http request. The web server answers by supplying the resource

- Most important HTTP requests (5 more less prominent)

GET	Requests a representation of the specified resource.	safe
PUT	Uploads a representation of the specified resource.	idempotent
DELETE	Deletes the specified resource.	idempotent
POST	Submits data to be processed (e.g., from a web form) to the identified resource.	

- **Definition 4.3.5** We call a HTTP request safe, iff it does not change the state in the web server. (except for server logs, counters,...; no side effects)
- **Definition 4.3.6** We call a HTTP request idempotent, iff executing it twice has the same effect as executing it once.
- HTTP is a stateless protocol (very memory-efficient for the server.)

Finally, we come to the last component, the web server, which is responsible for providing the web page requested by the user.

Web Servers

- **Definition 4.3.7** A web server is a network program that delivers web pages and supplementary resources to and receives content from user agents via the hypertext transfer protocol.
- **Example 4.3.8 (Common Web Servers)**
  - apache is an open source web server that serves about 60% of the WWWeb.
  - IIS is a proprietary server provided by Microsoft.
  - nginx is a lightweight open source web server.

▷ Even though web servers are very complex software systems, they come preinstalled on most UNIX systems and can be downloaded for Windows [XAM].

Now that we have seen all the components we fortify our intuition of what actually goes down the net by tracing the http messages.

Example: An http request in real life

▷ Connect to the web server (port 80) (so that we can see what is happening)

```
telnet www.kwarc.info 80
```

▷ Send off the GET request

```
GET /teaching/GenCS2.html http/1.1
Host: www.kwarc.info
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; en-US; rv:1.9.2.4)
  Gecko/20100413 Firefox/3.6.4
```

▷ Response from the server

```
HTTP/1.1 200 OK
Date: Mon, 03 May 2010 06:48:36 GMT
Server: Apache/2.2.9 (Debian) DAV/2 SVN/1.5.1 mod_fastcgi/2.4.6 PHP/5.2.6-1+lenny8 with
     Suhosin-Patch mod_python/3.3.1 Python/2.5.2 mod_ssl/2.2.9 OpenSSL/0.9.8g
Last-Modified: Sun, 02 May 2010 13:09:19 GMT
ETag: "1c78b-db1-4859c2f221dc0"
Accept-Ranges: bytes
Content-Length: 3505
Content-Type: text/html

<!--This file was generated by ws2html.xsl. Do NOT edit manually! -->
<html xmlns="http://www.w3.org/1999/xhtml"><head>...</head></html>
```


4.4 Multimedia Documents on the World Wide Web

We have seen the client-server infrastructure of the WWWeb, which essentially specifies how hypertext documents are retrieved. Now we look into the documents themselves.

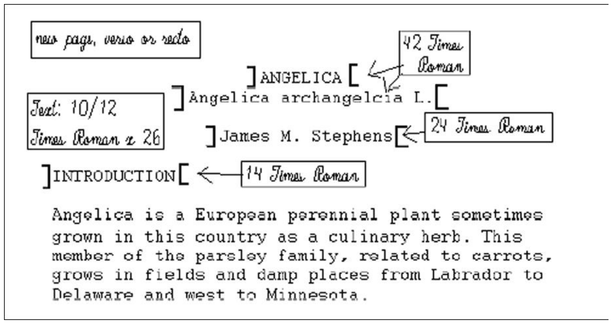
In ?character-encodings? have already discussed how texts can be encoded in files. But for the rich documents we see on the WWWeb, we have to realize that documents are more than just sequences of characters. This is traditionally captured in the notion of document markup.

Document Markup

▷ **Definition 4.4.1** Document markup is the process of adding markup codes to a document to control the structure, formatting, or the relationship among its parts.

▷ **Remark 4.4.2** Document markup turns plain text into formatted text.

▷ **Example 4.4.3** A text with markup codes (for printing)



There are many systems for document markup ranging from informal ones as in ?document-markup.ex? that specify the intended document appearance to humans – in this case the printer – to technical ones which can be understood by machines but serving the same purpose.

WWWeb documents have a specialized markuplanguage that mixes markup for document structure with layout markup, hyper-references, and interaction. The HTML markup elements always concern text fragments, they can be nested but may not otherwise overlap. This essentially turns a text into a document tree.

HTML: Hypertext Markup Language

- **Definition 4.4.4** The HyperText Markup Language (HTML), is a representation format for web pages. Version 4.01 is defined in [RHJ98].
- **Definition 4.4.5 (Main markup elements of HTML)** HTML marks up the structure and appearance of text with tags of the form `<el>` (begin tag), `</el>` (end tag), and `<el/>` (empty tag), where `el` is one of the following

structure	html,head, body	metadata	title, link, meta
headings	h1, h2, ..., h6	paragraphs	p, br
lists	ul, ol, dl, ..., li	hyperlinks	a
images	img	tables	table, th, tr, td, ...
styling	style, div, span	old style	b, u, tt, i, ...
interaction	script	forms	form, input, button

- **Example 4.4.6** A (very simple) HTML file with a single paragraph.

```
<html>
  <body>
    <p>Hello GenCS students!</p>
  </body>
</html>
```


The thing to understand here is that HTML uses the characters <, >, and / to delimit the markup. All markup is in the form of tags, so anything that is not between < and > is the textual content.

We will not introduce the various tags and elements of the HTML language here, but refer the reader to the HTML recommendation [RHJ98] and the plethora of excellent web tutorials.

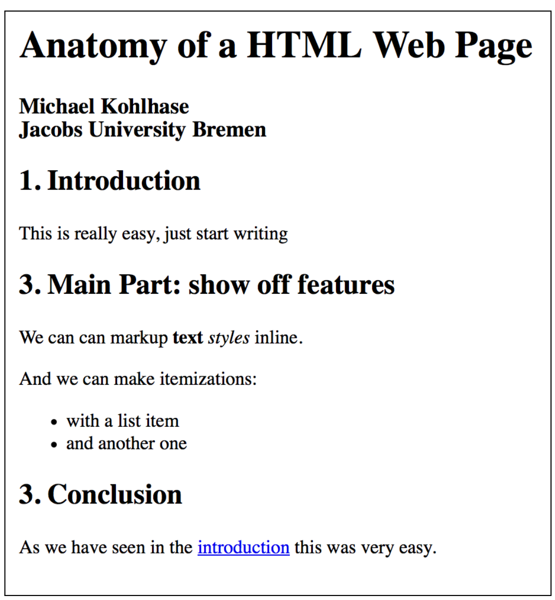
The best way to understand HTML is via an example. Here we have prepared a simple file that shows off some of the basic functionality of HTML.

A very first HTML Example (Source)

```
<html xmlns="http:www.w3.org/1999/xhtml">
  <head>
    <title>A first HTML Web Page</title>
  </head>
  <body>
    <h1>Anatomy of a HTML Web Page</h1>
    <h3>Michael Kohlhase<br/>Jacobs University Bremen</h3>
    <h2 id="intro">1. Introduction</h2>
    <p>This is really easy, just start writing.</p>
    <h2>3. Main Part: show off features</h2>
    <p>We can can markup <b>text</b> <em>styles</em> inline.</p>
    <p> And we can make itemizations:
      <ul>
        <li> with a list item</li>
        <li> and another one</li>
      </ul>
    </p>
    <h2>3. Conclusion</h2>
    <p> As we have seen in the <a href="#intro">introduction</a> this
    was very easy.</p>
  </body>
</html>
```


The thing to understand here is that HTML markup is itself a well-balanced structure of begin and end tags. That wrap other balanced HTML structures and – eventually – textual content. The HTML recommendation [RHJ98] specifies the visual appearance expectation and interactions afforded by the respective tags, which HTML-aware software systems – e.g. a web browser – then execute. In the next slide we see how `FireFox` displays the HTML document from the previous.

A very first HTML Example (Result)



Anatomy of a HTML Web Page

Michael Kohlhase
Jacobs University Bremen

1. Introduction

This is really easy, just start writing

3. Main Part: show off features

We can can markup **text** *styles* inline.

And we can make itemizations:

- with a list item
- and another one

3. Conclusion

As we have seen in the introduction this was very easy.

As the WWWeb evolved from a hypertext system purely aimed at human readers to an Web of multimedia documents, where machines perform added-value services like searching or aggregating, it became more important that machines could understand critical aspects web pages. One way to facilitate this is to separate markup that specifies the content and functionality from markup that specifies human-oriented layout and presentation (together called "styling"). This is what "cascading style sheets" set out to do. Another motivation for CSS is that we often want the styling of a web page to be customizable (e.g. for vision-impaired readers).

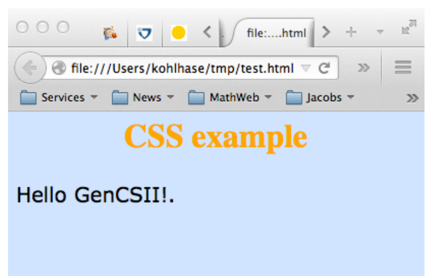
CSS: Cascading Style Sheets

▷ Idea: Separate structure/function from appearance.

Definition 4.4.7 The Cascading Style Sheets (CSS), is a style sheet language that allows authors and users to attach style (e.g., fonts and spacing) to structured documents. Current version 2.1 is defined in [BCHL09].

▷ **Example 4.4.8** Our text file from Example 4.4.6 with embedded CSS

```
<html>
  <head>
    <style type="text/css">
       body {background-color:#d0e4fe;}
       h1 {color:orange;
                 text-align:center;}
       p {font-family:"Verdana";
                 font-size:20px;}
     </style>
  </head>
  <body>
    <h1>CSS example</h1>
    <p>Hello GenCSII!.</p>
  </body>
</html>
```



Again, we explore this new technology by way of an example. We rework the title box from the HTML example above – after all treating author/affiliation information as headers is not very semantic. Here we use div and span elements, which are generic block-level (i.e. paragraph-like) an inline containers, which can be styled via CSS classes. The class titlebox is represented by the CSS selector .titlebox.

A Styled HTML Title Box (Source)

```
<head>
  <title>A Styled HTML Title</title>
  <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
  <div class="titlebox">
    <div class="title">Anatomy of a HTML Web Page</div>
    <div class="author">
      <span class="name">Michael Kohlhase</span>
      <span class="affil">Jacobs University Bremen</span>
    </div>
  </div>
  ...
```

```
.titlebox {border: 1px solid black;
           padding: 10px;
           text−align: center
           font−family: verdana;}
```

```
.title {font–size: 300%;
          font–weight: bold}
.author {font–size: 160%;
           font–style: italic;}
.affil {font–variant: small–caps;}
```


And here is the result:

A Styled HTML Title Box (Result)

Anatomy of a HTML Web Page

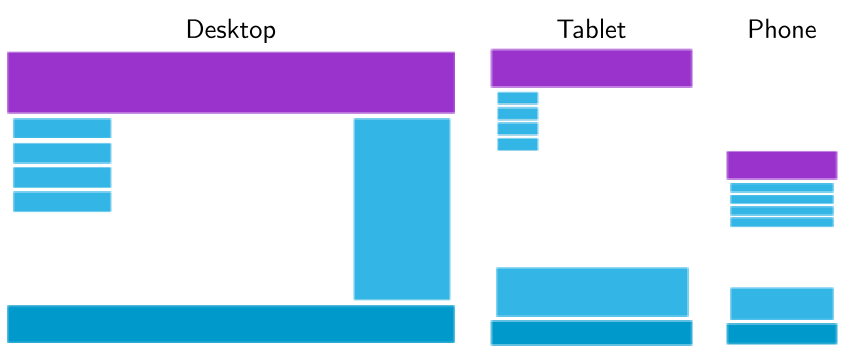
Michael Kohlhase JACOBS UNIVERSITY BREMEN

...

One of the important applications of the content/form separation made possible by CSS is to tailor webpage layout to the screen size and resolution of the device it is viewed on. Of course, it would be possible to maintain multiple layouts for a web page – one per screensize/resolution class, but a better way is to have one layout that changes according to the device context. This is what we will briefly look at now.

CSS Application: Responsive Design

- ▷ Problem: What is the screen size/resolution of my device?
- ▷ **Definition 4.4.9** Responsive web design (RWD) designs web documents so that they can be viewed with a minimum of resizing, panning, and scrolling – across a wide range of devices (from desktop computer monitors to mobile phones)
- ▷ **Example 4.4.10** web page with content blocks



Implementation: CSS-based layout with relative sizes and media queries– CSS conditionals based on client screen size/resolution/. . .

HTML was created in 1990 and standardized in version 4 in 1997. Since then there has HTML has been basically stable for more than a decade, even though in that time the WWWeb has evolved considerably from a web of static web pages to a Web in which highly dynamic web pages become user interfaces for web-based applications and even mobile applets. Acknowledging the growing discrepancy, the W3C has started the standardization of a successor version of HTML which has terminated with HTML5 in 2014.

▷ HTML5: The Next Generation HTML

▷ **Definition 4.4.11** The HyperText Markup Language (HTML5), is believed to be the next generation of HTML. It is defined by the W3C and the WhatWG.

▷ HTML5 includes support for (Details at [HBF+14])

- ▷ audio/video without plugins, (like the img tag earlier)
- ▷ a canvas element for scriptable, 2D, bitmapped graphics
- ▷ *SVG* for Scalable Vector Graphics
- ▷ MathML inline and display-style mathematical formulae

State of Play: All major browsers support HTML5 natively – except for MathML. Web content is slowly changing over.

We have seen a few different markup languages, and there are more to come. In this architecture, it is a problem to predict which markup language a given document is encoded, and thus how the systems should decode it.

▷ Specifying Document Types on the Web

▷ Problem: How to know how to decode/interpret a file fetched from the Web?

▷ Answer: Standardize "format identifiers" and integrate them into protocols.

▷ **Definition 4.4.12** A media type (also MIME type and content type) is a two-part identifier for file formats and format contents transmitted on the Internet. A media type is an ASCII string that adheres to the following grammar:

start	:==	toplevel '/' [tree] subtype[suffix][param]
toplevel	:==	'application' | 'audio' | 'example' | 'image' | 'message'
toplevel	:==	'model' | 'multipart' | 'text' | 'video'
tree	:==	iana | vnd | exp
suffix	:==	'+'ASCII
param	:==	';'ASCII
iana	:==	*IANA approved name*
vnd	:==	'vnd.' *IANA approved vendor/product name*
exp	:==	'x.' ASCII

▷ Media types are standardized and published by the Internet Assigned Numbers Authority (IANA).

▷ Media types first named MIME (Multipurpose Internet Mail Extensions) types, the name is still often used.

▷ **Example 4.4.13 (Common Media Types)**

`application/json`	JSON
`application/x-www-form-urlencoded`	URL encoded strings
`multipart/form-data`	form data from html form
`text/html`	HTML
`application/vnd.ms-powerpoint`	Microsoft Powerpoint
`application/epub+zip`	EPUB

1

[a] EdNote: add param and charset

4.5 Web Applications

In this Section we show how with a few additions to the basic WWWeb infrastructure introduced in Chapter 3, we can turn web pages into web-based applications that can be used without having to install additional software.

The first thing we need is a means to send information back to the web server, which can be used as input for the web application. Fortunately, this is already foreseen by the HTML format.

HTML Forms: Submitting Information to the Web Server

▷ **Example 4.5.1** Forms contain input fields and explanations.

```
<form name="input" action="submit.php" method="get">
  Username: <input type="text" name="user" />
  <input type="submit" value="Submit" />
</form>
```

The result is a form with three elements: a text, an input field, and a submit button, that will trigger a HTTP GET request to the URL specified in the `action` attribute.

As the WWWeb is based on a client-server architecture, computation in web applications can be executed either on the client (the web browser) or the server (the web server). For both we have a special technology; we start with computation on the web server.

4.5.1 Server Side Scripting

Server-Side Scripting: Programming Web Pages

▷ Idea: Why write HTML pages if we can also program them! (easy to do)

▷ **Definition 4.5.2** A server-side scripting framework is a web server extension that generates web pages upon HTTP GET requests.

▷ **Example 4.5.3** `perl` is a scripting language with good string manipulation facilities. `perl` CGI is an early server-side scripting framework based on this.

▷ Server-side scripting frameworks allow to make use of external resources (e.g. databases or data feeds) and computational services during web page generation.

▷ Problem: Most web page content is static (page head, text blocks, etc.) (and no HTML editing support in program editors)

▷ Idea: Embed program snippets into HTML pages. (only execute these, copy rest)

▷ **Definition 4.5.4** A server-side scripting language is a server side scripting framework where web pages are generated from HTML documents with embedded program fragments that are executed in context during web page generation.

▷ Note: No program code is left in the resulting web page after generation (important security concern)

To get a concrete intuition on the possibilities of server-side scripting frameworks, we will present PHP, a commonly used open source scripting framework. There are many other examples, but they mainly differ on syntax and advanced features.

PHP, a Server-Side Scripting Language

▷ **Definition 4.5.5** PHP (originally "Programmable Home Page Tools", later "PHP: Hypertext Processor") is a server-side scripting language with a C-like syntax. PHP code is embedded into HTML via special "tags" `<?php` and `?>`

▷ **Example 4.5.6** The following PHP program uses `echo` for string output

```
<html>
  <body><?php echo 'Hello world';?></body>
</html>
```

▷ **Example 4.5.7** We can access the server clock in PHP (and manipulate it)

```
<?php
$tomorrow = mktime(0,0,0,date("m"),date("d")+1,date("Y"));
echo "Tomorrow is ".date("d. m. Y", $tomorrow);
?>
This fragment inserts tomorrow's date into a web page
```

▷ **Example 4.5.8** We can generate pages from a database (here `MySQL`)

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
```

```
mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br />";
  }

mysql_close($con);
?>
```

▷ **Example 4.5.9** We can even send e-mail via this e-mail form.

```
<html><body>
<?php
if (isset($_REQUEST['email']))//if "email" is filled out, send email
  {//send email
  $email = $_REQUEST['email'] ;
  $subject = $_REQUEST['subject'] ;
  $message = $_REQUEST['message'] ;
  mail("someone@example.com", $subject,
  $message, "From:" . $email);
  echo "Thank you for using our mail form";}
else //if "email" is not filled out, display the form
  {echo "<form method='post' action='mailform.php'>
   Email: <input name='email' type='text' /><br />
   Subject: <input name='subject' type='text' /><br />
   Message:<br />
   <textarea name='message' rows='15' cols='40'>
   </textarea><br />
   <input type='submit' />
   </form>";}
?>
</body></html>
```


With server-side scripting frameworks like PHP, we can already build web applications, which we now define.

Web Applications: Using Applications without Installing

▷ **Definition 4.5.10** A web application is a website that serves as a user interface for a server-based application using a web browser as the client.

▷ **Example 4.5.11** Commonly used web applications include

- http://ebay.com; auction pages are generated from databases
- http://www.weather.com; weather information generated weather feeds
- http://slashdot.org; aggregation of news feeds/discussions
- http://github.com; source code hosting and project management

Common Traits: pages generated from databases and external feeds, content submission via HTML forms, file upload

▷ **Definition 4.5.12** A web application framework is a software framework for creating web applications.

▷ **Example 4.5.13** The LAMP stack is a web application framework based on linux, apache, MySQL, and PHP.

▷ **Example 4.5.14** A variant of the LAMP stack is available for `Windows` as XAMPP [XAM].

Indeed, the first web applications were essentially built in this way. Note however, that as we remarked above, no PHP code remains in the generated web pages, which thus "look like" static web pages to the client, even though they were generated dynamically on the server.

There is one problem however with web applications that is difficult to solve with the technologies so far. We want web applications to give the user a consistent user experience even though they are made up of multiple web pages. In a regular application we we only want to login once and expect the application to remember e.g. our username and password over the course of the various interactions with the system. For web applications this poses a technical problem which we now discuss.

State in Web Applications and Cookies

▷ Recall: Web applications contain multiple pages, HTTP is a stateless protocol.

▷ Problem: how do we pass state between pages? (e.g. username, password)

▷ Simple Solution: Pass information along in query part of page URLs.

▷ **Example 4.5.15 (HTTP GET for Single Login)** Since we are generating pages we can generated augmented links

```
<a href="http://example.org/more.html?user=joe,pass=hideme">... more</a>
```

Problem: only works for limited amounts of information and for a single session

▷ Other Solution: Store state persistently on the client hard disk

▷ **Definition 4.5.16** A cookie is a text file stored on the client hard disk by the web browser. Web servers can request the browser to store and send cookies.

▷ Note: cookies are data not programs, they do not generate pop-ups or behave like viruses, but they can include your log-in name and browser preferences.

▷ Note: cookies can be convenient, but they can be used to gather information about you and your browsing habits.

▷ **Definition 4.5.17** third party cookies are used by advertising companies to track users across multiple sites. (but you can turn off, and even delete cookies)

Note that that both solutions to the state problem are not ideal, for usernames and passwords the URL-based solution is particularly problematic, since HTTP transmits URLs in GET requests without encryption, and in our example passwords would be visible to anybody with a packet sniffer. Here cookies are little better as cookies, since they can be requested by any website you visit.

We now turn to client-side computation

4.5.2 Client-Side Computation

One of the main advantages of moving documents from their traditional ink-on-paper form into an electronic form is that we can interact with them more directly. But there are many more interactions than just browsing hyperlinks we can think of: adding margin notes, looking up definitions or translations of particular words, or copy-and-pasting mathematical formulae into a computer algebra system. All of them (and many more) can be made, if we make documents programmable. For that we need three ingredients: *i)* a machine-accessible representation of the document structure, and *ii)* a program interpreter in the web browser, and *iii)* a way to send programs to the browser together with the documents. We will sketch the WWWeb solution to this in the following.

To understand client-side computation, we first need to understand the way browsers render HTML pages.

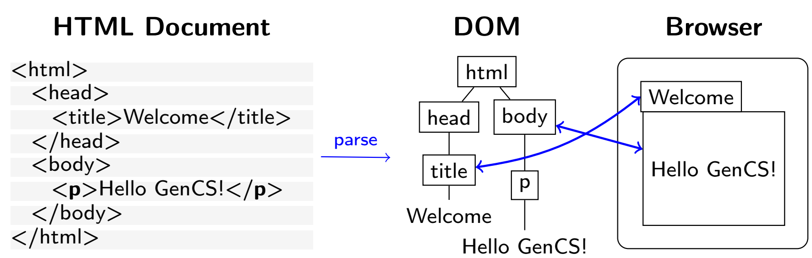
Background: Rendering Pipeline in Browsers

▷ Observation: The nested, markup codes turn HTML documents into trees.

▷ **Definition 4.5.18** The document object model (DOM) is a data structure for the HTML document tree together with a standardized set of access methods.

▷ Rendering Pipeline: Rendering a web page proceeds in three steps

1) the browser receives a HTML document,
2) parses it into an internal data structure, the DOM,
3) which is then painted to the screen. (repaint whenever DOM changes)



The DOM is notified of any user events (resizing, clicks, hover,...)

The most important concept to grasp here is the tight synchronization between the DOM and the screen. The DOM is first established by parsing (i.e. interpreting) the input, and is synchronized with with the browser UI and document viewport. As the DOM is persistant and synchronized, any change in the DOM is directly mirrored in the browser viewpoint, as a consequence we only need to change the DOM to change its presentation in the browser. This exactly the purpose of the client side scripting language, which we will go into next.

Dynamic HTML

▷ Idea: generate parts of the web page dynamically by manipulating the DOM.

▷ **Definition 4.5.19** JavaScript is an object-oriented scripting language mostly used to enable programmatic access to the DOM in a web browser.

▷ JavaScript is standardized by ECMA in [ECM09].

▷ **Example 4.5.20** We write the some text into a HTML document object (the document API)

```
<html>
 <head>
  <script type="text/javascript">document.write("Dynamic HTML!");</script>
 </head>
 <body><!-- nothing here; will be added by the script later --></body>
</html>
```


Let us fortify our intuition about dynamic HTML by going into a more involved example.

Applications and useful tricks in Dynamic HTML

▷ **Example 4.5.21** hide document parts by setting CSS `style` attributes to `display:none`

```
<html>
  <head>
    <style type="text/css">#dropper { display: none; }</style>
    <script language="JavaScript" type="text/javascript">
      window.onload = function toggleDiv(element){
           if(document.getElementById(element).style.display == 'none')
                {document.getElementById(element).style.display = 'block'}
           else if(document.getElementById(element).style.display == 'block')
                {document.getElementById(element).style.display = 'none'}}
     </script>
  </head>
  <body>
    <button onclick="toggleDiv('dropper')">...more </button>
    <div id="dropper"><p>Now you see it!</p></div>
  </body>
</html>
```

Application: write "gmail" or "google docs" as JavaScript enhanced web applications. (client-side computation for immediate reaction)

▷ Current Megatrend: Computation in the "cloud", browsers (or "apps") as user interfaces

Current web applications include simple office software (word processors, online spreadsheets, and presentation tools), but can also include more advanced applications such as project management, computer-aided design, video editing and point-of-sale. These are only possible if we carefully balance the effects of server-side and client-side computation. The former is needed for computational resources and data persistence (data can be stored on the server) and the latter to keep personal information near the user and react to local context (e.g. screen size).

Chapter 5

An Overview over XML Technologies

We have seen that many of the technologies that deal with marked-up documents utilize the tree-like structure of (the DOM) of HTML documents. Indeed, it is possible to abstract from the concrete vocabulary of HTML that the intended layout of hypertexts and the function of its fragments, and build a generic framework for document trees. This is what we will study in this Chapter.

Excursion: XML (EXtensible Markup Language)

- ▷ XML is language family for the Web
  - ▷ tree representation language (begin/end brackets)
  - ▷ restrict instances by *Doc. Type Def. (DTD)* or *Schema* (Grammar)
  - ▷ Presentation markup by *style files* (XSL: XML Style Language)

Intuition: XML is extensible HTML & simplified SGML

- ⊳⊳ logic annotation (*markup*) instead of presentation!
- ▷ many tools available: parsers, compression, data bases, ...
- ▷ conceptually: transfer of directed graphs instead of strings.
- ▷ details at http://www.w3c.org

The idea of XML being an "extensible" markup language may be a bit of a misnomer. It is made "extensible" by giving language designers ways of specifying their own vocabularies. As such XML does not have a vocabulary of its own, so we could have also it an "empty" markup language that can be filled with a vocabulary.

XML is Everywhere (E.g. document metadata)

- ▷ **Example 5.0.1** Open a PDF file in `Acrobat Reader`, then cklick on *File* ↘ *DocumentProperties* ↘ *Document* you get the following text: (showing only a small part)

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
         xmlns:iX='http://ns.adobe.com/iX/1.0/'>
  <rdf:Description xmlns:pdf='http://ns.adobe.com/pdf/1.3/'>
    <pdf:CreationDate>2004-09-08T16:14:07Z</pdf:CreationDate>
```

```
    <pdf:ModDate>2004-09-08T16:14:07Z</pdf:ModDate>
    <pdf:Producer>Acrobat Distiller 5.0 (Windows)</pdf:Producer>
    <pdf:Author>Herbert Jaeger</pdf:Author>
    <pdf:Creator>Acrobat PDFMaker 5.0 for Word</pdf:Creator>
    <pdf:Title>Exercises for ACS 1, Fall 2003</pdf:Title>
  </rdf:Description>
  ...
  <rdf:Description xmlns:dc='http://purl.org/dc/elements/1.1/'>
    <dc:creator>Herbert Jaeger</dc:creator>
    <dc:title>Exercises for ACS 1, Fall 2003</dc:title>
  </rdf:Description>
</rdf:RDF>
```


This is an excerpt from the document metadata which **Acrobat Distiller** saves along with each PDF document it creates. It contains various kinds of information about the creator of the document, its title, the software version used in creating it and much more. Document metadata is useful for libraries, bookselling companies, all kind of text databases, book search engines, and generally all institutions or persons or programs that wish to get an overview of some set of books, documents, texts. The important thing about this document metadata text is that it is not written in an arbitrary, PDF-proprietary format. Document metadata only make sense if these metadata are independent of the specific format of the text. The metadata that MS Word saves with each Word document should be in the same format as the metadata that Amazon saves with each of its book records, and again the same that the British library uses, etc.

XML is Everywhere (E.g. Web Pages)

▷ **Example 5.0.2** Open web page file in FireFox, then click on $View \searrow PageSource$, you get the following text: (showing only a small part and reformatting)

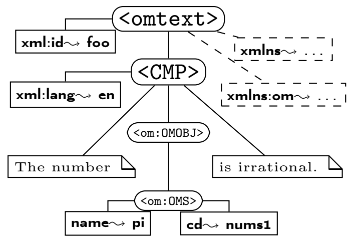
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Michael Kohlhase</title>
    <meta name="generator"
          content="Page generated from XML sources with the WSML package"/>
  </head>
  <body>...
    <p>
      <i>Professor of Computer Science</i><br/>
      Jacobs University<br/><br/>
      <strong>Mailing address - Jacobs (except Thursdays)</strong><br/>
      <a href="http://www.jacobs-university.de/schools/ses">
        School of Engineering &amp; Science
       </a><br/>...
    </p>...
  </body>
</html>
```

▷ **Definition 5.0.3** XHTML is the XML version of HTML(just make it valid XML)

XML Documents as Trees

▷ Idea: An XML Document is a Tree

```
<omtext xml:id="foo"
   xmlns="..."
   xmlns:om="...">
  <CMP xml:lang='en'>
   The number
   <om:OMOBJ>
     <om:OMS cd="nums1"
             name="pi"/>
   <om:OMOBJ>
   is irrational.
  </CMP>
</omtext>
```



- ▷ **Definition 5.0.4** The XML document tree is made up of element nodes, attribute nodes, text nodes (and namespace declarations, comments,. . .)
- ▷ **Definition 5.0.5** For communication this tree is serialized into a balanced bracketing structure, where
  - ▷ an element is represented by the brackets <el> (called the opening tag) and </el> (called the closing tag).
  - ▷ The leaves of the tree are represented by empty elements (serialized as <el></el>, which can be abbreviated as <el/>
  - ▷ and text nodes (serialized as a sequence of UniCode characters).
  - ▷ An element node can be annotated by further information using attribute nodes— serialized as an attribute in its opening tag

Note: As a document is a tree, the XML specification mandates that there must be a unique document root.

▷ The Dual Role of Grammar in XML (I)

- ▷ The XML specification [XML] contains a large character-level grammar. (81 productions)

NameChar :== Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender

Name :== (Letter | '_' | ':') $(\text{NameChar})^*$

element :== EmptyElementTag | STag content ETag

STag :== '<' $(S)^*$ Name $(S)^*$ attribute $(S)^*$ '>'

ETag :== '</' $(S)^*$ Name $(S)^*$ '>'

EmptyElementTag :== '<' $(S)^*$ Name $(S)^*$ attribute $(S)^*$ '/>'

- ▷ use these to parse well-formed XML document into a tree data structure
- ▷ use these to serialize a tree data structure into a well-formed XML document
- ▷ Idea: Integrate XML parsers/serializers into all programming languages to communicate trees instead of strings. (more structure $\hat{=}$ better CS)

The Dual Role of Grammar in XML (II)

- ▷ Idea: We can define our own XML language by defining our own elements and attributes.
- ▷ Validation: Specify your language with a tree grammar (works like a charm)
- ▷ **Definition 5.0.6** Document Type Definitions (DTDs) are grammars that are built into the XML framework.

  Put <DOCTYPE foo PUBLIC "foo.dtd">! into the second line of the document to validate.
- ▷ **Definition 5.0.7** RelaxNG is a modern XML grammar/schema framework on top of the XML framework.

RelaxNG, A tree Grammar for XML

- ▷ **Definition 5.0.8** RelaxNG (RelaxNG: Regular Language for XML Next Generation) is a tree grammar framework for XML documents.

  A RelaxNG schema is itself an XML document; however, RelaxNG also offers a popular, non-XML compact syntax.
- ▷ **Example 5.0.9** The RelaxNG grammars validate the left document

document	RelaxNG in XML	RelaxNG compact
<lecture> <slide id="foo"> first slide </slide> <slide id="bar"> second one </slide> </lecture>	<grammar> <start> <element name="lecture"> <oneOrMore> <ref name="slide"/> </oneOrMore> </element> </start> <define name="slide"> <element name="slide"> <text/> </element> <attribute name="id"> <text/> </attribute> </define> </grammar>	start = element lecture {slide+} slide = element slide {attribute id {text} text}

The Document Object Model

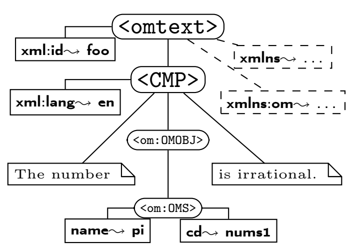
- ▷ **Definition 5.0.10** The document object model (DOM) is a data structure for storing documents as marked-up documents as document trees together with a standardized set of access methods for manipulating them.

One of the great advantages of viewing marked-up documents as trees is that we can describe subsets of its nodes.

XPath, A Language for talking about XML Tree Fragments

▷ **Definition 5.0.11** The XML path language (XPath) is a language framework for specifying fragments of XML trees.

▷ **Example 5.0.12**



XPath exp.	fragment
/	root
omtext/CMP/*	all <CMP> children
//@name	the name attribute on the <OMS> element
//CMP/*[1]	the first child of all <OMS> elements
//*[@cd='nums1']	all elements whose cd has value nums1

An XPath processor is an application or library that reads an XML file into a DOM and given an XPath expression returns (pointers to) the set of nodes in the DOM that satisfy the expression.

XSLT, A tree Transformer for XML

▷ **Definition 5.0.13** XSLT (Extensible Stylesheet Language Transformations) is a declarative, XML-based language used for the transformation of XML documents. It is standardized by the W3C.

▷ **Definition 5.0.14** XSLT stylesheets consist of a set of templates which match a XML elements via an XPath expression and create a result tree.

▷ **Definition 5.0.15** An XSLT Processor is a program that takes an XSLT stylesheet S and an XML file X as input and transforms X as specified by the templates in S.

▷ **Example 5.0.16** There are various open source or free XSLT processors

- ▷ xsltproc [Vei] is very fast, but only supports XSLT version 1.
- ▷ saxon [Kay08] supports XSLT version 2, but is slower.

▷ **Example 5.0.17** Use this stylesheet to extract a numbered table of contents from an HTML document

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html><body><xsl:apply-templates select="//h1"/></body></html>
```

```
  </xsl:template>

  <xsl:template match="*"/>

  <xsl:template match="h1">
    <p style="font-size:large">
       <xsl:value-of select="count(preceeding-sibling::h1)"/>
       <xsl:text>. </xsl:text>
       <xsl:copy-of select="*|text()"/>
    </p>
  </xsl:template>
</xsl:stylesheet>
```

Chapter 6

Electronic Books and their Formats

We will now come to a technology for reading books that is becoming ever more important: eletronic books – they accounted for almost a quarter of book sales in the US in 2015, and half that worldwide. In this Chapter we will mainly take a detailed look at the EPUB format which is a good example for a standard that builds on other established document standards but tailors them to a specific application.

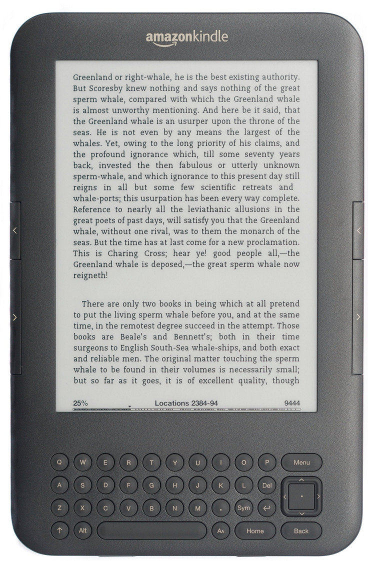
Electronic Books

▷ **Definition 6.0.1** An electronic book (eBook) is a publication in electronic form that can be read on digital devices.

▷ **Example 6.0.2** Arguably the first eBooks were the texts provided by Project Gutenberg in 1971. [GO]

▷ **Definition 6.0.3** An electronic book reader (eReader) is a hardware or software device for reading electronic books.

▷ **Example 6.0.4** Popular hardware-based eReaders are Kindle (Amazon.com), the iPad (Apple), and the Nook (Barnes&Noble), but software readers also abound.



There are quite a few representation formats for electronic books, here we will cover the most important open one: EPUB; it is also one of the earliest formats.

EPUB: A Standard for Electronic Publishing

▷ **Definition 6.0.5** EPUB is a free and open standard [GMG] for electronic books provided by the International Digital Publishing Forum (IDPF) [IDP]
It consists of three specifications:

1) The "EPUB Content Documents" [GMCE], profiles of HTML5 and CSS, (and restricted JavaScript) for the document contents.
2) The "EPUB Container Format" (OCF), which describes the structure of the EPUB file in XML and collects all files as a ZIP archive.
3) The "EPUB Media Overlays" [DW], for synchronising text and audio.

The current version is 3.0.1 (v3.1 under development)

▷ EPUB files usually have the extension `.epub3`.

▷ EPUB does not specify a format for digital rights management (DRM), which makes it less attractive for the big publishers.

▷ EPUB is supported by almost all eReaders and publishing software.

The EPUB format heavily leverages existing web standards and has over time considerably adapted to accomodate new versions. Often to the effect that original EPUB technologies are superseded by new ones that become available in the included standards. This "subsidiarity principle" greatly facilitates adoption and implementation of the standard.

But not all of the things the included standards allow are sensible for electronic books. Therefore EPUB defines profiles– restrictions that single out the meaningful documents for them. For instance, the allowed use of JavaScript is very restricted in EPUB, after all, electronic books should behave like "books", not like applications.

To validate an EPUB document use e.g. the IPDF validator [EV].

We will now go into the Open Container Format and show concrete examples for an ebook.

EPUB: Open Container Format

▷ **Definition 6.0.6** An EPUB file is a group of files wrapped in a ZIP file. The Open Container Format (OCF) specifies how these files should be organized [PG].

▷ The `mimetype` file must be a text document in `ASCII` and must contain the EPUB media type `application/epub+zip`. It must also be uncompressed, unencrypted, and the first file in the ZIP archive.

▷ The purpose of this file is to provide a more reliable way for applications to identify the mimetype of the file than just the `.epub3` extension.

▷ Also, there must be a folder named `META-INF` which contains the required file `container.xml`.

▷ **Definition 6.0.7** The container file `container.xml` is XML specifies the root files of the eBook , which specifies the package and its rendering as a book-like structure.

The OCF format is the heart of the EPUB specification, which is mostly about packaging HTML5 content files. The ZIP format ensures that we obtain a single file, so that distribution becomes simple. Other document formats like ODF, the "Open Document format" (`OpenOffice`/`LibreOffice`) or the "Office Open XML" (`MS Word`) do the same, even the JAR files for compiled JAVA programs do.

The `mimetype` and `container.xml` files are for identiftying the contents of the package independently of the file extension.

An Example Container

ZIP Container

```
mimetype
META-INF/
  container.xml
book.opf
nav.xhtml
chapter1.xhtml
chapter2.xhtml
...
ch1-pic.png
style.css
myfont.otf
```

`container.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<container version="1.0"
    xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <rootfiles>
    <rootfile full-path="book.opf"
        media-type="application/oebps-package+xml"/>
  </rootfiles>
</container>
```


The example above is a simple book (the first book in the "Lord of the Rings"), where we have a couple of HTML5 files for the chapters of the book, a picture (the map of Middle Earch) some CSS style information, and a special font (for the elvish runes).

The actual contents and rendiering of an eBook are described in the package document, which is specified as the root file in the container file. Actually, there could be multiple root files. Each one specifies a possible rendering of the book.

EPUB: Package Documents

▷ **Definition 6.0.8** A Package Documents specify additional structure and coherence to an electronic book in EPUB. It specifies the

- ▷ ebook contents (what files) in the `manifest` element
- ▷ metadata (author, date, etc.) in the `metadata` element
- ▷ linear reading order in the `spine` element, and
- ▷ (optionally) important structural components in the `guide` element.

of the package. Package documents are identified by the namespace `http://www.idpf.org/2007/opf`, the media type `application/oebps-package+xml`, and the file extension `.opf`.

▷ **Definition 6.0.9** The navigation control of the an EPUB gives a machine-readable table of contents of the book in HTML5.

The following package document specifies the contents and reading order for the "Fellowship of the rings". We have a "manifest" (after a "cargo manifes" listing the cargo, passengers, and crew of a ship, aircraft, or vehicle), which lists, identifies, and specifies the media types of all files in the package and specifies the linear reading order.

An Example EPUB Package Document

```
<?xml version="1.0"?>
<package version="2.0" xmlns="http://www.idpf.org/2007/opf" unique-identifier="BookId">

  <metadata xmlns:dc="http://purl.org/dc/elements/1.1/"
                   xmlns:opf="http://www.idpf.org/2007/opf">
    <dc:title>The Fellowship of the Ring</dc:title>
    <dc:language>en</dc:language>
    <dc:identifier id="BookId" opf:scheme="ISBN">9780007117116</dc:identifier>
    <dc:creator opf:file-as="Tolkien, JRR" opf:role="aut">JRR Tolkien</dc:creator>
  </metadata>

  <manifest>
    <item id="chapter1" href="chapter1.xhtml" media-type="application/xhtml+xml"/>
    <item id="chapter2" href="chapter2.xhtml" media-type="application/xhtml+xml"/>
    <item id="chapter3" href="chapter3.xhtml" media-type="application/xhtml+xml"/>
    <item id="stylesheet" href="style.css" media-type="text/css"/>
    <item id="ch1-pic" href="ch1-pic.png" media-type="image/png"/>
    <item id="myfont" href="css/myfont.otf" media-type="application/x-font-opentype"/>
    <item id="toc" href="nav.xhtml" media-type="application/xhtml+xml" properties="nav"/>
  </manifest>

  <spine toc="ncx">
    <itemref idref="toc"/>
    <itemref idref="chapter1" />
    <itemref idref="chapter2" />
    <itemref idref="chapter3" />
  </spine>

  <guide>
    <reference type="loi" title="List Of Illustrations" href="appendix.html#figures" />
  </guide>

</package>
```


Finally, we have the table of contents, for which we just use HTML5 again. This has a `nav` element for blocks of navigation links. As this is exactly what we need,

An Example Navigation Control file

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ops="http://www.idpf.org/2007/ops"
      xml:lang="de">
  <head>
    <title>Table of Contents</title>
  </head>
  <body>
   <nav ops:type="toc">
    <h1>The Fellowship of the Ring</h1>

    <h2>Book One</h2>
    <ol>
      <li><a href="nav.xhtml">Table of Contents</a></li>
      <li><a href="chapter1.xhtml">A Long-expected Party</a></li>
      <li><a href="chapter2.xhtml">The Shadow of the Past</a></li>
      <li><a href="chapter3.xhtml">Three is Company </a></li>
      <li>...</li>
    </ol>
    <h2>Book Two</h2>
    ...
```

```
  </nav>
 </body>
</html>
```


Even though there are still many, many more facets to the EPUB format, this little introduction should be enough to build eletronic books in EPUB from scratch.

Part III

Computing with Text Documents

Chapter 7

Revision Control and Project Planning Systems

We address a very important topic for document management: supporting the document life-cycle as a collaborative process. In this Chapter we discuss how we can use a set of tools that have been developed for supporting collaborative development of large program collections can be used for document management.

We will first introduce the problems and attempts at solutions and the introduce two classes of revision control systems and discuss their paradigmatic systems.

7.1 Dealing with Large/Distributed Projects and Document Collections

In this Section we will look at problems in managing the artefacts of large projects that create some kind of document collection. Such projects range from technical documentation for complex systems over knowledge collections like the Wikipedia, to software like the Linux kernel. They have in common that a *large group of authors/developers* manage a *large document collection* over a *long period of time*.

Large/Distributed Document Collections

- ▷ **Observation 7.1.1** *Document Collections can get large and long-lived*
- ▷ Problem: How to manage them effectively?
- ▷ **Example 7.1.2** We will use the following projects/systems as running examples and characterize them by size.
  - ▷ The "Subversion Book" [CSFP04] (ca. 450 pages, 9 translations, 3 main authors, hundreds of contributors, since 2002)
  - ▷ linux kernel (ca. 16m lines of code, ca. 12 000 contributors, since 1991),
  - ▷ wikipedia ($\geq$ 5m articles, $\geq$ 280 languages, ca. 40 m files, $\geq$ 130 k active users, since 2001).
  - ▷ "2048": a simple browser/app game with lots and lots of variants (forks) in three years.

The first is a relatively standard book about a revision control system (see below), while the wikipedia and linux kernel are paradigmatic examples of a large document collections and software development. The last example was chosen as an example of a population of program variants
EdN:2 that develop together, exchanging code and ideas as they evolve. [2]

For most of the examples above it is clear that the document collections are ever-changing; after all that is their ultimate purpose. But even for documents that we perceive as rather static (e.g. novels) there is a "document lifecycle" – if only before it is published.

Lifecycle Management for Digital Documents (Technical Book)

- ▷ Documents may have a non-trivial life-cycle involving multiple actors.
- ▷ **Example 7.1.3** For any book we have the following stages:
  1) skeleton/layout (chapters, characters, interactions)
  2) first complete draft (given out to test readers)
  3) private editing cycle ⇝ accepted draft (testing with more readers, refining/condensing the story)
  4) publisher's editing cycle ⇝ final draft (professional editor proposes refinements to the draft)
  5) copyediting for spelling, adherence of publisher's house style
  6) adding artwork/cover ⇝ first published edition
  7) e-dition (eBook) etc. (different artwork, links, interactivity)
- ▷ **Example 7.1.4** For technical books, multiple editions follow to adapt them to changing domain or correct errors.

As the document lifecycle problems are common to all document collections, various solutions and practices have evolved to cope with them. We will briefly present and evaluate them in the following. For all them the critial question is how they deal with multiple files and multiple/distributed authors/developers – a single author/developer working on a single file can usually cope quite well. Multiple variants of the document collections – e.g. in different languages or variants of the domain further complicate matters and mandate system support.

The first practice of collaborating on a document is probably the most widespread: multiple authors collaborate on a single document – or very a limited number of documents and distribute the respective newest state to their collaborators. Some word processors have support for tracking changes, which may help in the process. Even though the version information could in principle be looked up in the document metadata, it is common practice to add the current date and the last author in the file date.

Document Lifecycle Mgmt. & Collaboration Approaches

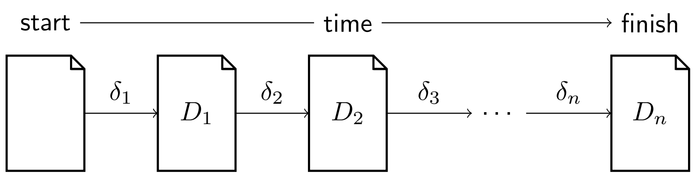
- ▷ Practice: Send around MS Word documents by e-mail (dates in file name)

[2] EdNote: MI: I used those as running examples to explains concepts of branching, merging, snapshots, reverting, etc. and why they are so important and come up all the time — thats also how I explained the lifecycle graph for revision control systems.

▷ Characteristics/Problems:

++ well-understood technology (no training need)
– version tracking as a social process (error prone)
– merging diverging versions is annoying (manual process)
– archiving past versions optional/manual (storage problems)
– no multifile support, no snapshots

▷ Summary: only supports serial collaboration, no multifile support



larger teams ⇝ more time wasted

The main problem in this practice is that if two – or more – authors change the document in different ways, we say that the document diverges, someone must merge the variants to get to a common state again – a tedious undertaking at best without machine support. The solution to this problem is to socially enforce a linear development timeline: "if you make an iteration until tomorrow morning, then I can take over until noon, . . . ".

Instead of distributing the documents to the collaborators we can also upload the respective version to a central server which keeps the respective "current version" for download by the collaborators.

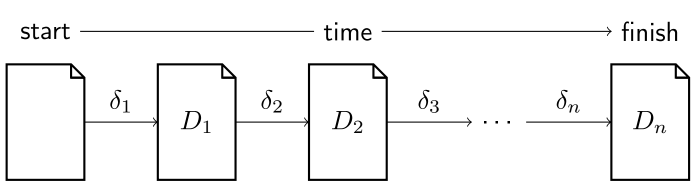
Document Lifecycle Mgmt. & Collaboration Approaches

▷ Practice: Put your documents on Dropbox or MS Sharepoint, or use a Wiki.

▷ Characteristics/Problems:

– local install of (proprietary) software
+ auto-synchronization between cloud and user copies upon save
+ auto-archiving past versions in cloud
– merging diverging versions unsupported (manual process)
– no multifile support, no snapshots

▷ Summary: only supports serial collaboration



larger teams ⇝ more time wasted

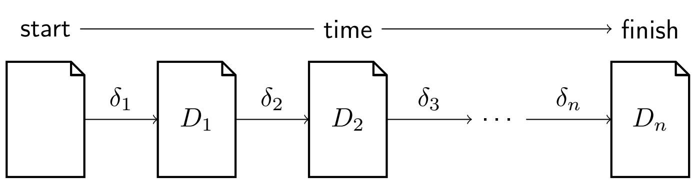
A central server immediately solves the problem of identifying the "current version", and usually also provides date/time of the last change and the author of that change. A server also enforces a linear development. On a naive server later uploads overwrite previous ones. To remedy this, more advanced servers give the authors access to old versions of documents. This is in fact very important, since it may be necessary to revert certain changes, e.g. to reinstate inadvertent deletions.

While a history-aware server (Dropbox and MS Sharepoint are) allows for a non-linear multi-file development path in principle, system support for this is missing.

The next practice is somewhat complementary from the last, even though it is technically a radical extension: changes are uploaded to the server and merged into the document character-by-character. In particular, this guarantees a linear timeline and a consistent document state.

Document Lifecycle Mgmt. & Collaboration Approaches

- ▷ Practice: Use real-time collaborative editors like `EtherPad` or wordprocessors like `GoogleDocs` or `Office Online`.
- ▷ Characteristics/Problems:
  - + browser-based, no installation necessary
  - + real-time auto-synchronization between cloud and user copies
  - +− extremely detailed auto-archiving past versions in cloud
  - − no diverging versions
  - − no multifile support, no snapshots
- ▷ Summary: only supports serial collaboration



larger teams $\leadsto$ more time wasted

While automatic document consistency is directly guaranteed by the system, intra document, semantic consistency is very hard to achieve, as there is usually no possibility to block out other authors in order to do a larger rewrite. Though the systems give access to the version history, it's character-by-character nature makes it very difficult to spot useful versions.

It is a general observation that while real-time collaborative editing is very convenient and effective for single small documents, where semantic intra- and inter-document consistency plays an subordinate role, it does not scale to large document collections and author collectives.

The last practice in collaborative document lifecycle management is to use a revision control system. These systems were originally built for managing the lifecycles of large software projects with multiple, distributed developer groups and even more individual files. As a consequence, they answer all the shortcomings of the practices we have reviewed above, but are restricted to text files – as programs tend to be.

Document Lifecycle Mgmt. & Collaboration Approaches

▷ Practice: Use revision control system (good for ASCII-based file formats)

▷ Characteristics/Problems:

– special install, training necessary

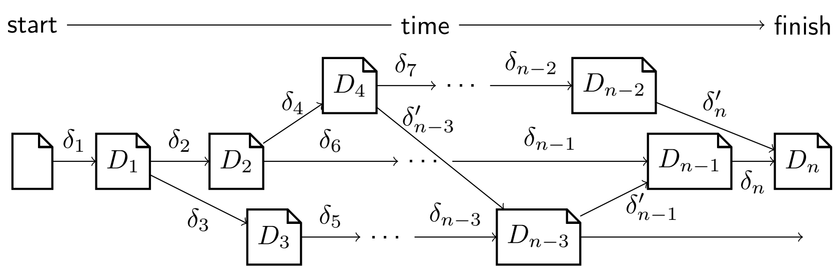
- optimized for character/line-based formats

+ user-initiated synchronization between cloud and user copies

+ auto-archiving past versions on server

++ multifile support, snapshots, merging support, tagging

▷ Summary: supports parallel, branching collaboration



larger teams ⇝ large-scale parallelization/experimentation

The main idea behind such systems is that we can manage very large document collections and author collectives by making the "document collection changes" – expressed by δ in the figure above – the prime objects in our system. Changes can be passed around, applied to working copies, and merged – if we restrict ourselves to text files.

If we look at the paradigmatic document collections from our motivation, then we see that Wikipedia uses the "central server" solution – it is based on a wiki server, while all the others use version control systems.

We will now take a closer look at revision control systems and how they work. Following a somewhat historic path, we will first look at a paradigmatic centralized revision control systems and then advance to the currently dominant distributed system, building on the concepts introduced for the centralized system.

7.2 Centralized Version Control

We start out with the basics of revision control system based on a relatively simple architecture with a central repository with which all developers interact.

Revision Control Systems

▷ **Definition 7.2.1** A revision control system is a software system that tracks the change process of a document collection via a federation of repositories that store the development history of the collection. Each step in the development history is called a revision.

▷ **Definition 7.2.2** Users do not directly work on the repository, but on a working copy that is synchronized with the repository by revision control actions

1) checkout: creates a new working copy from the repository
2) update: merges the differences between the revision of the working copy and the revision of the repository into the working copy.
3) commit: transmits the differences between the repository revision and the working copy to the repository, which registers them, patches the repository revision, and makes this the new repository revision – called the head revision or simply the head.

▷ **Observation 7.2.3** *The commits determine the revisionss in a revision control system.*

Remark: revision control systems usually store the head revision explicitly and can compute development histories via reverse diffs.

Definition 7.2.1 and Definition 7.2.2 are very general, so that they can cover a wide variety of architectures.

Before we become more concrete, let us have a look at the basic ingredient of revision control systems: computing differences, applying them to documents, and reconciling differences.

▷ Computing and Managing Differences with diff & patch

▷ **Definition 7.2.4** diff is a file comparison utility that computes differences between two text files f_1 and f_2. Differences are output linewise in a diff file (also called a patch), which can be applied to f_1 to obtain f_2 via the patch utility.

▷ **Example 7.2.5**

`The quick brown`
`fox jumps over`
`the lazy dog`
	`The quack brown`
	
	`fox jumps over`
	`the loozy dog`
		`1c1,2`
		`< The quick brown`
		`---`
		`> The quack brown`
		`>`
		`3c4`
		`< the lazy dog`
		`---`
		`> the loozy dog`

▷ **Definition 7.2.6** A diff file consists of a sequence of hunks that in turn consist of a locator which contrasts the source and target locations (in terms of line numbers) followed by the added/deleted lines.

Merging Differences with merge3

▷ There are basically two ways of merge the differences of files into one.

▷ **Definition 7.2.7** In two-way merge, an automated procedure tries to combine two different files by copying over differences by guessing or asking the user.

▷ **Definition 7.2.8** In three-way merge the files are assumed to be created by changing a joint original (the parent) by editing. The merge3 tool examines the differences and patterns appearing in the changes between both files as well as the parent, building a relationship model to generate a new revision. Usually, non-conflicting differences (affecting only one of the files) can directly be copied over.

 85 JACOBS UNIVERSITY

With this, we can now understand the revision control worfklows in our concrete system.

In its simplest form, a revision control system, can be understood using the Subversion system that is heavily used in open source projects that have a relatively hierarchical development model.

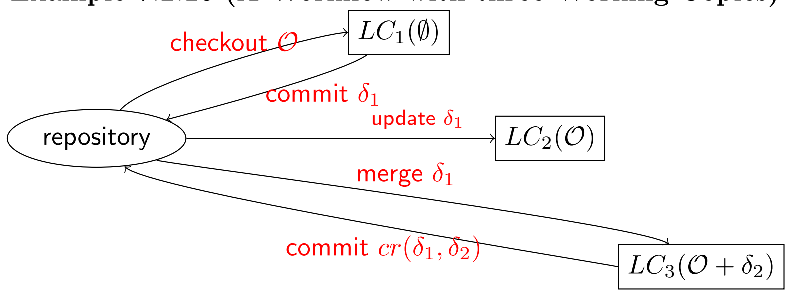
Centralized Version Control (with Subversion)

▷ **Definition 7.2.9** Subversion is a centralized revision control system that features

- ▷ a single, central repository (for current revision and reverse diffs)
- ▷ local working copies (asynchronous checkouts, updates, commits)

They are kept synchronized by passing around diff differences and patching the repository and working copies. Conflicts are resolved by (three-way) merge.

▷ **Example 7.2.10 (A Workflow with three Working Copies)**



 86 JACOBS UNIVERSITY

In the workflow of Example 7.2.10 is a typical one:

1) A first user checks out a new working copy LC_1, from the empty repository, adds a couple of files – we denote the new document collection at this point with $\mathcal{O}$, and commits the difference δ_1 between the working copy and $\mathcal{O}$ to the repository which δ_1 logs it as "revision 1".

2) There is another repository LC_2, which has been checked out earlier (i.e. based on "revision 0"), and which is now no longer in sync with the repository. So we can update (i.e. patch)

it to "revision 1" by transferring δ_1 to LC_2, which thus has same content as LC_1, namely $\mathcal{O}$.

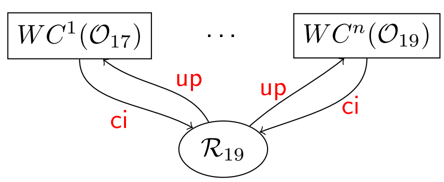
3) For a third repository LC_3 which has been checked out at "revision 0" we assume that it has been changed by adding different files, the difference being δ_2. Note that as these changes are relative to "revision 0", they cannot simply be committed to the repository. Therefore we need to update it. As LC_3 already contains changes, this amounts to a merge of δ_1 and δ_2 to get a new local copy that is essentially $\mathcal{O} + \delta 2$, which is now relative to "revision 1". This can now be committed to the repository to form "revision 2".

Note: that in all of this it does not matter who the authors of the respective changes and the owners of the respective working copies are. They might be different persons, or a single author might have multiple working copies, e.g. one one the work computer, one on a laptop, and one on the home desktop. They are all held in sync by updates, commits.

With this basic mechanism, we can already model quite complex and collaborative workflows. The basic idea is simple: we just use the update/commit cycle to synchronize a set of working copies.

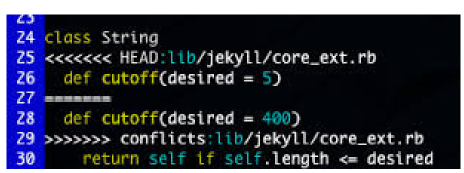
Collaboration with Subversion

- Idea: We can use the same technique for collaboration between multiple working copies.
- Diff-Based Collaboration:



The Subversion system takes care of the synchronization:

- you can only commit, if your revision is HEAD (otherwise update)
- update merges the changes into your working copy
- If there are changes on the same line, you have a conflict.



Note: that these collaborative workflows can be asynchronous. In particular working copies can lag behind the repository as long as they want – they only have to synchronize for commits. This gives a lot of freedom in the development process.

Also note: that unless the repository and the working copies are on the same computer – which is somewhat unlikely. Commits and updates are only possible while online, this sometimes prevents authors/developers from grouping changes logically as they have to collect them until they are online again.

Subversion even allows to update to a specific revision, e.g. if an author wants to base her work on that – or wants to revert some changes[1]. In fact, Subversion supports branching: committing different development lines to the repository, but we will not go into this here and leave the discussion for later when we discuss distributed revision control systems where branching is the main mechanism of operation.

Branching: Supporting Multiple Lines of Development

- ▷ **Observation 7.2.11** *A central repository entails – ultimately – a single line of development. ⇝ changes have to be merged into the repository eventually.*
- ▷ But: we want to develop – and commit – to variants in parallel. w
- ▷ **Definition 7.2.12** A branch is a copy of an object under revision control (such as a source code file or a directory tree) so that it can be developed in parallel.
- ▷ In particular, branches allow parallel development histories via separate commits.
- ▷ commits from one branch can be merged into another.
- ▷ **Example 7.2.13** In software development we profit from separate
  - ▷ master branch/trunk– main line of development, used for integration.
  - ▷ release branch– only bug fixing; no new features
  - ▷ feature branch– develop a new feature; close branch upon merge
  - ▷ staging branch– integrate multiple fixes/features
- ▷ **Definition 7.2.14** A branch controlled by a different developer or not intended to be merged back is called a fork.

Branches are easy to realize in the diff/patch/merge-based architecture.

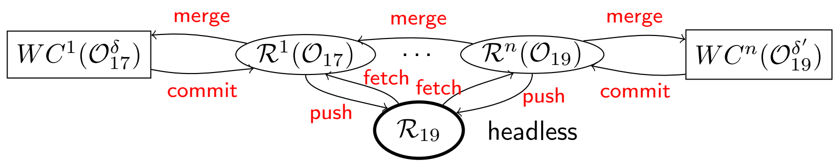
7.3 Distributed Revision Control

In this Section we will introduce distributed revision control systems using the git system as an example. As this is the currently dominant system, we will also go into more detail about concrete usage of the system.

Distributed Version Control

- ▷ Problems with Centralized Revision Control (Subversion):
  1) we can only commit when online!
  2) all collaboration goes via one, central repository. (prescribes workflow)
- ▷ Idea: Distribute the repositories and move patches between them.

[1] Don't drink and write!



1) local commits to local repositories

2) all repositories created equal (flexible organization)

▷ **Definition 7.3.1** We call a revision control system distributed, iff it allows multiple repositories that can exchanged patches. Contrastingly we call a revision control system centralized, if it only allows one repository.

▷ **Definition 7.3.2** We call a repository headless (or bare), if used without working copy.

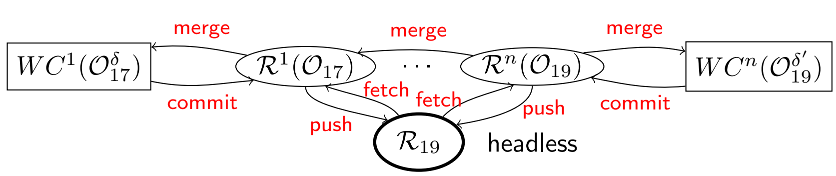
The concept of distributed revision control systems is motivated by the two shortcomings at the top of the slide, which can be remedies by a single – if relatively radical idea: allowing lots of repositories that can communicate with each other by exchanging patches. Local repositories allow commits while offline and distributed repositories allow for flexible architectures.

Of course, there is a price to pay: instead of having three main revision control actions we now have five. We need to be able to move commits to a remote repository and fetch commits from one. This makes the model quite a lot more complicated.

Centralized vs. Distributed Version Control

▷ Intuition: Distributed revision control systems generalize centralized ones.

Centralized	Distributed	Centralized	Distributed
repository	headless repository	commit	commit + push
working copy	repository + working copy	update	fetch + merge
		checkout	fetch + checkout



Distributed Version Control with git

▷ **Definition 7.3.3** git is a distributed revision control system that features

- ▹ local repositories (contains head and reverse diffs)
- ▹ local working copies (local commits)

- ▷ multiple remote repositories (branches/forks)
- ▷ local changes can pushed to a remote repository
- ▷ changes from a remote repository can be pulled into the local one.

▷ **Definition 7.3.4** There are various repository management systems that facilitate providing repositories, e.g.

- ▷ GitHub, a repository hosting service at `http://GitHub.com` (free public repositories)
- ▷ GitLab, an open source repository management system (http://gitlab.org)

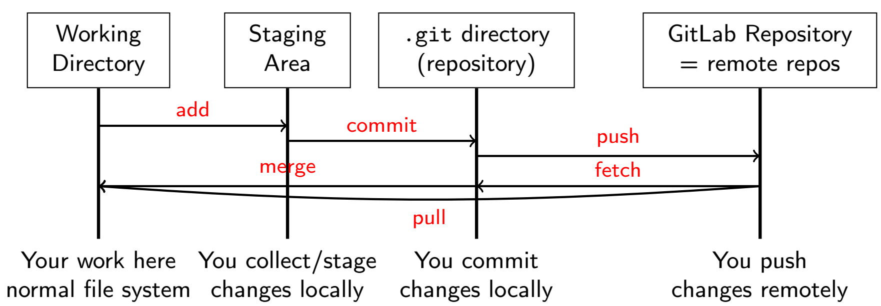
Now that we understand the concepts, let us see how we can use them in practice.

For this we assume that students have installed git on their computers, so that they can use it; [CS14, section 1.5] gives an excellent introduction. In all of our concrete examples, we will use `UNIX` shell commands; for `Windows` users should use the git shell, a git-enhanced version of the `UNIX` shell, and *not* the `Windows` command prompt.

Working with git

▷ Use the shell and shell commands for simplicity/clarity (originally `UNIX`)

▷ Overview: git local workflow: staging changes



commits act only on staged files ⇝ `git add foo.tex`

▷ basic git commands (there are many variants and options ⇝ study them)

`git clone ⟪URI⟫`	clones the repos at ⟪URI⟫
`git add ⟪file⟫`	stages ⟪file⟫
`git commit -m'⟪msg⟫'`	commits staged files with commit message ⟪msg⟫
`git push ⟪repos⟫ ⟪branch⟫`	pushes all commits to branch ⟪branch⟫ on ⟪repos⟫
`git pull ⟪repos⟫ ⟪branch⟫`	fetches and merges branch ⟪branch⟫ from ⟪repos⟫
`git status`	gives information about the working copy.

We have only shown the most basic commands here. There are many other commands an options that make your life much easier. Before you start, you should configure some global options for git (just adapt the following lines and type them into the shell).

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

The following two lines configure `git` to always pull the branch called `master` from the repository called `origin`

```
$ git config branch.master.remote origin
$ git config branch.master.merge refs/heads/master
```

With this configuration you can replace `git push origin master` with a simple `git push`.

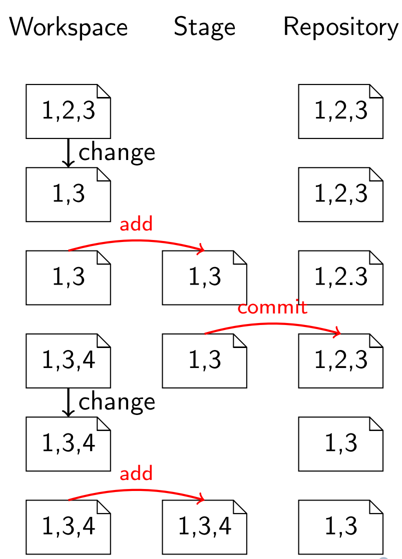
Finally, the `-a` option is very useful for `git commit`: it automatically stages all the changed files. `git commit -am'foo'` commits all your change in the current directory (which is often what you want).

An Example

```
> git init
Initialized empty Git repository in /tmp
> echo "1,2,3" > test.txt
> git add test.txt
> git commit −am'initializing'

> echo "1,3" > test.txt
> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update ...
  (use "git checkout −− <file>..." to...
        modified: test.txt
no changes added to commit
(use "git add" and/or "git commit −a")

> git add test.txt
> git commit −m'bla' test.txt
> echo "1,3,4" > test.txt
> git add test.txt
```



We can now come back to the topic, where git really shines: branch branching. The main reason for this is that merging is so well-supported in git. Together with the distributed "local-repository" architecture, this allows for very flexible organization of workflows. We will discuss the basics of branch-based and fork-based workflows here.

git Branches, Remote Repositories

▷ git special commands for making, switching, and merging branches.

`git branch` ⟪branch⟫	makes a branch with name ⟪name⟫
`git checkout` ⟪branch⟫	switches a working copy to branch ⟪branch⟫
`git branch -v`	shows all branches
`git branch -d` ⟪branch⟫	deletes branch ⟪branch⟫

Intuition: In git branches are very similar to repositories, but more lightweight.

Repositories can have different permissions.

⊳ Fork-based Collaboration: If you want to contribute to a repository $\mathcal{R}$ you have no push-rights on,

1) clone $\mathcal{R}$ to a new repository $\mathcal{R}'$ you own (i.e. fork it; $\mathcal{R}'$ is a fork of $\mathcal{R}$)
2) develop your contribution on $\mathcal{R}'$.
3) ask $\mathcal{R}$s owners to pull from $\mathcal{R}'$ (pull request)

git repository management systems like GitHub and GitLab support this.

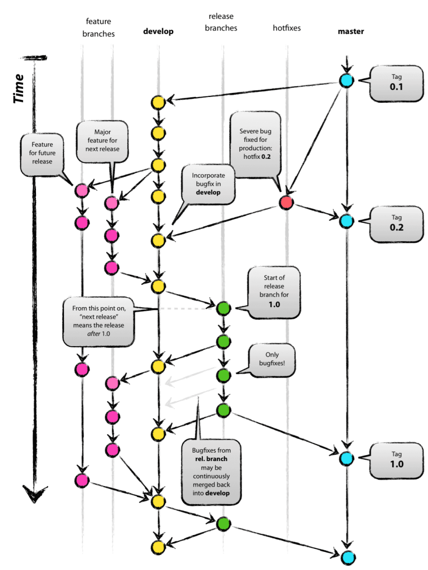
▷ Git commands for working with remote repositories:

`git remote add` ⟪name⟫ ⟪URI⟫	gives the repos at ⟪URI⟫ the name ⟪name⟫
`git remote show`	shows all remote repositories

What we have seen above, let us briefly disuss an elaborate workfow suitable for large development teams, which has become known under the name "GitFlow".

GitFlow: An Elaborate Development Model based on GIT

▷ [Dri10] suggests a development model with feature branches, ...



We will now complement revision control systems, as discussed above, with issue tracking systems. The former support dealing with changes in the collaborative development of document collections, the latter support the collaboratie management of issues – the reasons for changes.

7.4 Bug/Issue Tracking Systems

In this Section we will discuss issue tracking systems, which support the collaborative management of reports on a particular problem, its status, and other relevant data. These systems originated from tracking systems for help desks and in software engineering, but have evolved into general project planning systems. We will mainly look at systems that originate from software engineering applications here.

Bug/Issue Tracking Systems

- **Definition 7.4.1** An issue tracker (also called issue tracking system simply bugtracker) is a software application that keeps track of reported issues– i.e. software bugs and feature requests – in software development projects.
- **Example 7.4.2** There are many open-source and commercial bugtrackers
  - bugzilla: `http://bugzilla.org` (Mozilla's bugtracker)
  - TRAC: `http://trac.edgewall.org` (mostly for Subversion)
  - GitHub: `http://github.com`
  - GitLab: `http://gitlab.com` (open source version of GitHub)
  - JIRA: `https://www.atlassian.com/software/jira` (proprietary)

  Most bugtrackers also integrate a wiki and integrate a revision control system via extended markdown.

Issue trackers manage issues and track their status over its whole lifetime – from the initial report to its resolution. This results in a particular set of components that are present in all systems.

▷ The Anatomy of an Issue

- **Definition 7.4.3** An issue (or bug report) specifies
  - title: a short and descriptive overview (one line)
  - description: a precise description of the expected and actual behavior, giving exact reference to the component, version, and environment in which the bug occurs. (bugs must be reproducible and localizable)
  - issue metadata: who, when, what, why, state, ... (see below)
  - discussion about the bug.
  - attachment: e.g. a screen shot, set of inputs, etc.

Issues – How to Write a Good One

- The descriptions or issues should be concise, but describe all pertinent aspects of the situation leading to the unexpected behavior
- **Example 7.4.4 (A bad bug report description)**
  My browser crashed. I think I was on foo.com. I think that this is a really bad

problem and you should fix it or else nobody will use your browser.

▷ **Example 7.4.5 (A good one)**
I crash each time I go to foo.com (Mozilla build 20000609, Win NT 4.0SP5). This link will crash Mozilla reproducibly unless you remove the `border=0` *attribute:*

```
<IMG SRC="http://foo.com/topicfoos.gif" width=34 border=0 alt="News">
```

Remember: developers are also human (try to minimize their work)

▷ **Definition 7.4.6** A feature request is an issue that only specifies the expected behavior and proposes ways of implementing that.

Markdown a simple Markup Language for Generating HTML.

▷ Idea: We can translate between markup languages.

▷ **Definition 7.4.7** Markdown is a family of markup languages whose contro words are unobtrusive and easy to write in a texteditor. It is intended to be converted to HTML and other formats for display.

▷ **Example 7.4.8** Markdown is used in applications that want to make user input easy and effective, e.g. wikis and issue tracking systems.

▷ Workflow: Users write markdown, which is formatted to HTML and then served for display.

▷ **Example 7.4.9**

Markdown syntax	Generated HTML

```
Heading
=======

Sub-heading
-----------

### Another deeper heading

Paragraphs are separated
by a blank line.

Two spaces at the end of a
line leave a line break.

Text attributes _italic_,
**bold**, `monospace`.

Bullet list:
  * apples
  * oranges
  * pears

Numbered list:
  1. apples
  2. oranges
  3. pears

A [link](http://example.com).
```

```
<h1>Heading</h1>
<h2>Sub-heading</h2>
<h3>Another deeper heading</h3>
<p>Paragraphs are separated
by a blank line.</p>
<p>Two spaces at the end of a
  line leave a<br/> line break.</p>
<p>Text attributes <em>italic</em>,
<strong>bold</strong>,
<code>monospace</code>.</p>
<p>Bullet list:</p>
<ul>
<li>apples</li>
<li>oranges</li>
<li>pears</li>
</ul>
<p>Numbered list:</p>
<ol>
<li>apples</li>
<li>oranges</li>
<li>pears</li>
</ol>
<p>A <a href="http://example.com">
   link</a>.</p>
```

Tracker-Specific Markdown Extensions

▷ **Remark 7.4.10** Source code hosting systems offer special extensions for referencing their components.

▷ **Example 7.4.11** GitLab recognizes

- ▷ `@foo` for team members (`@all` for all project members), e.g. *cc: `@miko`*
- ▷ #123 for issues, e.g. *depends on #4711*
- ▷ `!123` for merge requests, e.g. *but merge #19 first*
- ▷ $123 for code snippets, e.g. *see $123 for an example usage*
- ▷ 1234567 for commits, e.g. *fixed by `4c0decb` yesterday*.
- ▷ `[file](path/to/file)` for file references, e.g. *as we see in `[pre.tex](../lib/pre.tex)`*

▷ **Observation 7.4.12** *very useful for project planning and reporting*

Bugtracker Workflow

▷ Typical Workflow: supported by all bugtrackers

- ▷ user reports issue (files report in the system)
- ▷ other users extend/discuss/up/downvote issue
- ▷ QA engineer triages issues – classification, remove duplicates, identify dependencies, tie to component, ...
- ▷ developer accepts or re-assigns issue (fixes who is responsible primarily)
- ▷ project planning by identification of sub-issues, dependencies (new issues)
- ▷ bug fixing (design, implementation, testing)
- ▷ issue landing (sign-off, integration into code base)
- ▷ release of the fix (in the next revision)
- ▷ bug closure

▷ **Observation 7.4.13** *An issue tracker can serve as a full-blown project planning system, if used accordingly.*

▷ **Definition 7.4.14** For timing work plans, most bugtracker issuestracker provide milestoness that issues can be targetted to.

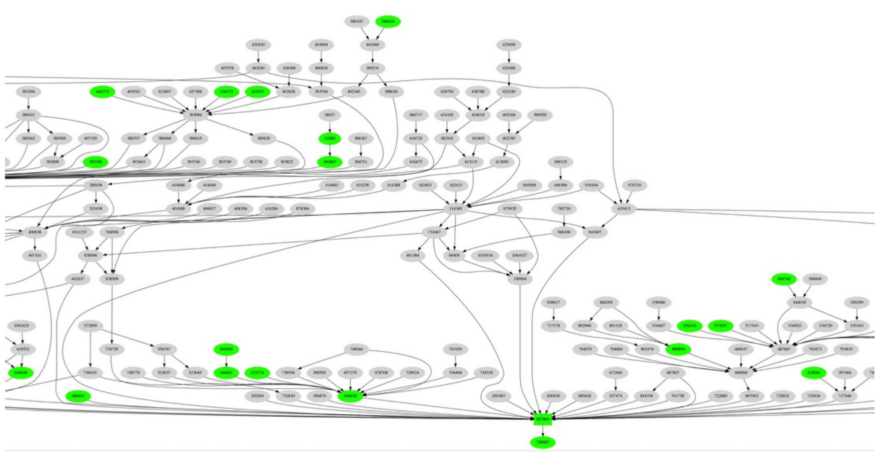
Administrative Metadata for Issues

▷ to make the issue-based workflows work we need data

▷ **Definition 7.4.15 (Administrative Metadata)** issue metadata can specify

- ▷ issue number: for referencing with e.g. #15
- ▷ an assignee: a developer currently responsible
- ▷ comments: a discussion thread focused on this issue.
- ▷ participants: people who get notified of changes/comments
- ▷ labels: for specializing bug search
- ▷ a status: e.g. one of `new`, `assigned`, `fixed/closed`, `reopened`.
- ▷ a resolution for fixed bugs, e.g.
  - ▷ FIXED: source updated and tested
  - ▷ INVALID: not a bug in the code
  - ▷ WONTFIX: "feature", not a bug
  - ▷ DUPLICATE: already reported elsewhere; include reference
  - ▷ WORKSFORME: couldnâĂŹt reproduce issue
- ▷ dependencies: which issues does this one depend on/block?

Dependency Graph of a Firefox Issue in Bugzilla



Chapter 8

Computing with Documents

There are several dialects of regular expression languages that differ in details, but share the general setup and syntax. Here we introduce the UNIX variant.

Regular Expressions, see [RE]

▷ **Definition 8.0.1** A regular expression (also called regexp) is a formal expression that specifies a set of strings.

▷ **Definition 8.0.2 (Meta-Characters for Regexps)**

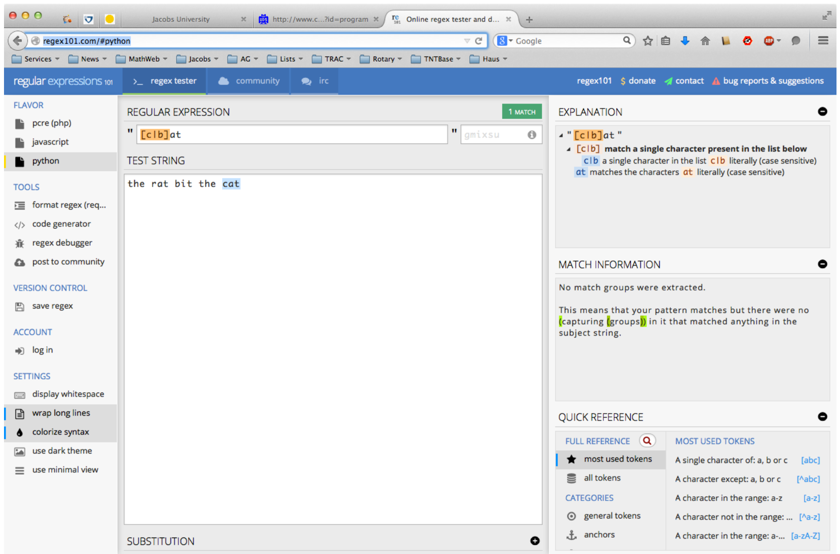
char	denotes
.	any single character
^	beginning of a string
$	end of a string
[...]	any single character in the brackets
[^...]	any single character not in the brackets
(...)	marks a group
\\n	the n^{th} group
|	disjunction
*	matches the preceding element zero or more times
+	matches the preceding element one or more times
?	matches the preceding element zero or one times
{n,m}	matches the preceding element between n and m times
\s	whitespace character
\S	non-whitespace character

All other characters match themselves, to match e.g. a ?, escape with a \: \?.

As we have seen regular expressions can become quite cryptic and long (cf. e.g. ?complex-regexp.ex?), so we need help in developing them. One way is to use one of the many regexp testers online

Playing with Regular Expressions

▷ If you want to play with regexps, go e.g. to http://regex101.com



The `sed` stream editor is an example of a standalone utility – it is shipped with most operating systems – that uses regular expressions. It can be used to automate repetitive editing operations on files.

The sed Stream Editor

- ▷ **Definition 8.0.3** The sed utility is a stream editor, it takes a stream (think file) and some regexp replacement commands as an input and gives a stream as a output.
- ▷ **Example 8.0.4** A sed command is of the form
  - ▷ s/⟪regexp⟫/⟪replacement⟫/ (replace once), or
  - ▷ s/⟪regexp⟫/⟪replacement⟫/g (replace globally).
- ▷ To invoke sed in a shell (e.g. on `linux`, `Mac OS X`, or `cygwin` on `Windows`)

```
sed -e 's/oldstuff/newstuff/g' inputFileName > outputFileName
```

or (if sedfile.**sed** contains many sed commands)

```
sed -f sedfile.sed inputFileName > outputFileName
```

- ▷ **Example 8.0.5 (Update the Jacobs Web Site)**

```
sed -e 's/International Univ/Jacobs Univ/g;s/IUB/Jacobs/g' index.html > index.html
```

- ▷ **Example 8.0.6 (Stalin eliminates Trotzki)** Let cleanse.**sed** be the sed file

```
s/Leon Trotzki//g;s/Trotzki//g
s/Lev Davidovich Bronstein//g;s/Davidovich//g;s/Bronstein//g
```

then Stalin can just use the following shell script to cleanse Kreml documents

```
find / −name −E ".*\.html|.*\.txt" −exec 'sed −f cleanse.sed {} > {} \;
```


Example 8.0.6 shows the power of `sed` in combination with other utilities. Here we use the UNIX `find` utility that searches a file system for files with certain characteristics – here file names that match the regexp `.*\.html.*txt|` and executes the `sed` script `cleanse` we defined earlier.

The `lex/flex` Lexer Generator

▷ **Definition 8.0.7** The `lex` is a generator of lexical analyzers (lexers), i.e. a program that reads a lexer specification and outputs C code for a lexer.

A lexer specification is a list of pairs $\langle R, P\rangle$, where R is a regexp and P is C code to be executed when R is matched.

`lex` is part of UNIX (proprietary), it is extended by the open-source `flex`.

▷ **Example 8.0.8 (Spotting Integers)**

```
-?[1-9][0-9]* {printf("Saw an integer: %s\n", yytext)}
.|\n { /* Ignore all other characters. */ }
```

If this input is given to `flex`, it will be converted into a C file, `lex.yy.c`. This can be compiled into an executable which matches and outputs strings of integers. For example, given the input abc123z.&*2ghj-6! the program will print:

```
Saw an integer: 123
Saw an integer: 2
Saw an integer: -6
```


`lex` Example: Tokenizing Artithmetic Expressions

▷ **Example 8.0.9** We want to build a simple calculator, so we need a tokenizer for arithmetic expressions. Here is the `flex` code for one (see [Vol11] for details):

```
delim          [ \t]
whitesp        {delim}+
digit          [0-9]
number         [-]?{digit}*[.]?{digit}+
%%
{number}  { sscanf(yytext, "%lf", &yylval); return NUMBER;}
"+"       { return PLUS; }
"-"       { return MINUS; }
"/"       { return SLASH; }
"*"       { return ASTERISK; }
"("       { return LPAREN; }
")"       { return RPAREN; }
"\n"      { return NEWLINE; }
{whitesp} { /* No action and no return */}
```

▷ The declarations before the %% are abbreviations for number (note that they are recursive)

▷ instead of printing notifications we just return token types (values are in yytext)

 108 JACOBS UNIVERSITY

The yacc/bison Parser Generator

▷ **Definition 8.0.10** yacc (Yet Another Compiler Compiler) is a parser generator, i.e. a program that reads a parser specification and outputs C code for a parser. Historically, yacc was used to generate the C parser in UNIX, today, it is superseded by open-source extensions, e.g. bison.

A yacc parser specification consists of three parts divided by %%.

1) token definitions that specify which tokens to expect from flex
2) grammar and the actions: $$ is the constructed result.
3) more C code, including the usual main function.

 109 JACOBS UNIVERSITY

yacc/bison Example: Building a Calculator

▷ **Example 8.0.11** We want to build a simple calculator, so we need a tokenizer for arithmetic expressions. Here is the yacc code for one (see [Vol11] for details):

```
 %token NEWLINE NUMBER PLUS MINUS SLASH ASTERISK LPAREN RPAREN
%%
input:                  /* empty string */
     | input line;
 line: NEWLINE
     | expr NEWLINE           { printf("\t%.10g\n",$1); };
expr: expr PLUS term         { $$ = $1 + $3; }
     | expr MINUS term        { $$ = $1 - $3; }
     | term;
term: term ASTERISK factor   { $$ = $1 * $3; }
     | term SLASH factor      { $$ = $1 / $3; }
     | factor;
factor:  LPAREN expr RPAREN  { $$ = $2; }
       | NUMBER;
%%
int main(void) {yyparse();exit(0)}
```

Using this to generate a parser with bison gives a program tcalc which is a simple calculator

```
-1.1 + 2 * ( 4 / 3 )
        1566666667
2+2
        4
```

 110 JACOBS UNIVERSITY

The perl Programming Language

▷ **Definition 8.0.12** `perl` is a high-level, general-purpose, interpreted, dynamic programming language that makes extensive use of regular expressions.

▷ `perl` can directly use `sed` commands (with more regexps and execute subroutines)

▷ instead of specifying the language, let us go through an example!

`perl` Example: Correcting and Anonymizing Documents

▷ **Example 8.0.13** We write an a program that makes simple corrections on documents and also crosses out all names.

- ▷ *The worst president of the US,arguably was George W. Bush, right?*
- ▷ *However,are you famILIar with Paul Erdős or Henri Poincaré?*(Unicode)

Here is the program:

- ▷ we first initialize and load modules

```
#!/usr/bin/perl -w
use warnings;
use utf8;
use Encode;
```

- ▷ then we decode the argument and put it into a variable

```
my $expr = shift;
$expr = decode('utf8',$expr);
```

- ▷ We put put a space after a comma,

```
$expr =~ s/,(\S)/, $1/g;
```

- ▷ next we make abbreviations for regular expressions to save space

```
$c=qr/\p{UpperCase_Letter}/;
$l=qr/\p{Lowercase_Letter}/;
```

- ▷ capitalize the first letter of a new sentence,

```
$expr =~ s/([?.!])\s($l)/$1." ".uc($2)/eg;
```

- ▷ remove capital letters in the middle of words

```
$expr =~ s/($l)($c+)($l)/$1.lc($2).$3/eg;
```

- ▷ and we cross-out for official public versions of government documents,

```
$expr =~ s/($c$l+ ($c$l*(\.?) )?$c$l+)/'X' x length($1)/eg;
```

- ▷ finally, we print the result

```
print $expr,"\n";
```

The worst president of the US,arguably was George W. Bush. right? becomes
The worst president of the US, arguably was XXXXXX XX XXXX right?

Chapter 9

Programming Documents

Idea: Even though documents should be thought of as sequences of characters with markup (and images, formulae, tables, etc.), we can also think of them as *programs that produce such characters with markup*. In some situations, this is profitable, e.g. when the documents have parts that can be computed from the rest, e.g. a table of contents, the section numberings, or indices. In such situations, the author does not need to type in the computable document fragments, but can just represent them by a command. A conversion program interprets such a "document program" (usually text interspersed with commands), executes all the commands, and outputs a document (without commands), which can then be read. The main advantage of the "documents as programs" paradigm is that the computed document fragments can never get out of sync with the rest of the document, which eases the maintenance burden over the document life-cycle.

There are various implementations of this idea, in this Chapter we present the TEX/LATEX system, in which the `pdflatex` program is used to transform documents with macros into `PDF`. Systems like PHP do similar things for the Web.

The TEX Typesetting System

▷ **Definition 9.0.1** Typesetting is the process of creating the visual appearance of a document by assembling glyphs (visual representations of characters; also called types) on pages.

▷ Since Gutenberg's time (to ca. 1975), typesetting was done by assembling movable types (special metal positives of single letters) into lines and later into pages, which were inked and the printed; or using negatives to form cast-metal positives for printing.

▷ **Definition 9.0.2** TEX is a typesetting program designed by Donald Knuth in 1978. It combines movable types (character boxes) with macro programming.

▷ **Definition 9.0.3** The `pdftex` program reads a file of text marked up with TEX macros and outputs `PDF`.

▷ **Example 9.0.4 (Hello World in TEX)** `pdftex` typesets the following TEX file

```
Hello, World \bye
```

The command sequence `\bye` stops `pdftex` and is not shown in the output.

Note that the "document program"

```
Hello, World \bye
```

the `pdftex` interprets all characters as "self-inserting characters", i.e the character "a" is essentially a command that inserts a character "a" into the PDF (in the right font and size).

We have already seen one document program command used by TEX above, and there are many more. Most of them insert special characters into the document or change the formatting. But TEX goes much further, it allows the author to define commands as well. This makes the TEX format self-extensible, and into a very expressive special purpose programming language for documents.

TEX Macros for Programming Documents

- ▷ TEX uses command sequences (words starting with "\"; also called macros) for special effects.
- ▷ **Example 9.0.5** `\bye` stops the formatter, `\alpha` prints α, `\int` prints $\int$,...
- ▷ Users can also define TEX macros as abbreviations via `\def`
- ▷ **Example 9.0.6** `\def\tdm{Text and Digital Media}` defines the macro `\tdm`.
  `We love the USC ‘‘\tdm’’!` expands to
  "We love the USC "Text and Digital Media"!
- ▷ TEX macros can have arguments specify with `#1`, `#2`. . . : delimit with `{` and `}`
- ▷ **Example 9.0.7** with the macro `\def\tnwhat#1{Text and \textbf{#1}}` `\tnwhat{Beer}` expands to "Text and **Beer**"

TEX was invented by a mathematician, so it is not a surprise that it is the most capable tool for typesetting formulae — an art that only a select few professional typesetters (humans who put lead into rows) could do.

Mathematical Formulae in TEX

- ▷ **Definition 9.0.8** TEX has a math mode for formulae delimited with `$` (inline math) or `\[` and `\]` (display math)
- ▷ **Example 9.0.9** Some TEX commands can be used everywhere: e.g. the Greek letters, `\alpha` prints α, `\beta` prints β,...
- ▷ **Example 9.0.10** Many TEX commands only make sense in math mode: e.g. superscripts with `^`, e.g. `x^3` gives x^3, subscripts with `_`, e.g. `x_{ij}` gives x_{ij}, `\int` prints $\int$, `\frac{1}{2}` prints $\frac{1}{2}$,...
- ▷ **Example 9.0.11** `$\int_0^\infty f(\theta) d\theta$` expands to $\int_0^\infty f(\theta)d\theta$
- ▷ **Example 9.0.12** Use macros in math mode as well: `\def\frac#1#2{#1\over #2}`
  Then `\[1+\frac{2}{2+\frac{3}{3+\ldots}}\]` expands to

$$1+\frac{2}{2+\frac{3}{3+\ldots}}$$

One of the things that TeX is useful for is to automate numbering of sections, subsections, footnotes, etc. For that TeX offers some basic data structures. Here we introduce counters, and show how we can make simple sectioning macros from them.

TeX Counters

- TeX uses special macros as counters, `\newcount`, allocates a counter, `\advance` alters it, and `\the` references it.
- **Example 9.0.13** We define a sectioning macros

```
\newcount\seccount % allocate a new counter for sections
\newcount\subseccount % allocate a new counter subsections
\seccount0\subseccount0 % initialise both with 0
\def\section#1{ % begin macro definition
\advance\seccount by 1 % step the counter
\subseccount0 % reset the subsection counter
\textbf{\Large\the\seccount. #1} % section number and title
} % end macro definition
\def\subsection#1{\advance\subseccount by 1
\textbf{\large\the\seccount.\the\subseccount. #1}}
```


Anyone who is experienced in programming realizes that TeX is not a modern programming language. But of course, it was conceived in 1978, the age of COBOL, and a lot has happened in programming language design since then. But even if it is relatively inconvenient and ugly code, it gets the job done.

We will now present a couple of internal macros that build up to more document automation that shows the advantages of programming documents: a serial letter macro.

TeX Conditionals

- TeX provides some conditionals for your use:
  e.g. `\ifx` compares two macros, `\ifnum` compares two number, and `\ifmmode` tells you if you are in math mode.
  `\if⟨⟨cond⟩⟩...\else...\fi` uses it.
- TeX uses special macros for user-defined conditionals, `\newif\if⟨⟨cond⟩⟩`, allocates a conditional, `⟨⟨cond⟩⟩true` and `⟨⟨cond⟩⟩false` alter it,

Programming a Chain Letter

- **Example 9.0.14 (A Parametric Reminder)**

```
\def\reminder#1#2{\hfill Bremen, \today\par\bigskip
\noindent Dear #1,\par\medskip\noindent
please be sure that you will not forget to come to the lecture
today. We are planning big things.\par\medskip\noindent
```

```
Sincerely,\par\bigskip\noindent #2\newpage}
```

▷ **Example 9.0.15 (Programming a Serial Letter)**
We can use arbitrary characters to delineate arguments in macro definitions.

```
\def\sletter#1,#2;{\def\first{#1}\def\second{#2}\def\empty{}
\ifx\first\empty\else\reminder{#1}{Thomas \& Michael}
\ifx\second\empty\else\sletter#2,;\fi\fi}
\def\serialletter#1{\sletter #1;}
```

Also nothing prevents us from using recursion.

▷ **Example 9.0.16 (Making a Serial Letter)**

```
\serialletter{Mati, Anca, Isabel, Calin}
```


Our serial letter example shows that with a bit of programming effort the self-extensibility of TEX can be used to automate various document-oriented tasks, or style the documents for a given situation. Naturally, this brought forth a vibrant community that started swapping and re-using TEX programs.

TEX Macro Packages

▷ Idea: Separate out common macro definitions into a separate file and include that via \input. (So we can reuse them over multiple documents)

▷ Actually: many people have already done that.

▷ The AMS (American Mathematical Society) supplies AMSTEX: TEX macros that make it more convenient to write Math (e.g. the \frac macro)

▷ Till Tantau supplies tikz (TEX ist kein Zeichenprogram): TEX macros that allow you to draw images.

▷ Leslie Lamport supplies LATEX, a set of TEX packages and classes. `pdflatex` is `pdftex` with the LATEX package macros pre-loaded.

▷ The bibTEX package handles bibliographic references.

The most widely used macro package for TEX is LATEX, there are tens of thousands of macro packages that use the basic LATEX infrastructure. LATEX is the standard for high-end document formatting for scientific/technical documents nowadays. We now show a typical document as model for your own documents.

The Anatomy of a LATEX Document

▷ **Example 9.0.17** A LATEX file: `main.tex`

```
\documentclass{article} % use the article class (Journal Article)
\title{Anatomy of a {\LaTeX} Document} % specify the title,
\author{Michael Kohlhase\\Jacobs University Bremen} % author,
\date{\today} % and date
\begin{document} % start the document
\maketitle % make the title
\tableofcontents % make the table of contents
\section{Introduction}\label{sec:intro}
This is really easy, just start writing,
\section{Main Part}\label{sec:main}
We refer the reader to~\cite{Lamport:ladps94} for details.
But there should be at least one formula:
\[1+\frac{2}{2+\frac{3}{3+\ldots}}\]
\section{Conclusion}\label{concl:intro}
As we already said in Section~\ref{sec:intro} on
p. \pageref{sec:intro} this was not so bad was it?
\bibliographystyle{alpha}
\bibliography{example}
\end{document}
```

▷ Format it with `pdflatex main` (generates `main.aux` for references)

and the bibTEX database used in it

▷ **Example 9.0.18** a bibTEX file `example.bib`

```
@BOOK{Lamport:ladps94,
  title = {LaTeX: A Document Preparation System, 2/e},
  publisher = {Addison Wesley},
  year = {1994},
  author = {Leslie Lamport}}
```

▷ Generate bibliography with `bibtex main`(it knows about `example.bib` from `main.aux`)

▷ run `pdflatex` twice (to get all the cross-references right)

The Result (generated parts in red)

Anatomy of a $\LaTeX$ Document

Michael Kohlhase
Jacobs University Bremen
April 20, 2016

Contents

1. Introduction

This is really easy, just start writing,

2. Main Part

We refer the reader to [Lam84] for details. But there should be at least one formula:

$$1 + \frac{2}{2 + \frac{3}{3 + \ldots}}$$

3. Conclusion

As we already said in Section 1 on p. 1 this was not so bad was it?

References

[Lam94] Leslie Lamport, *LaTeX: A Document Preparation System, 2/e*, Addison Wesley, 1994.

Chapter 10

Writing Technical Documentation and Manuals

10.1 Technical Documentation in DocBook

DocBook

- ▷ **Definition 10.1.1** DocBook is a content markup language for technical documentation based on SGML or XML. It supplies elements/tags for the logical of book-like documents.
- ▷ DocBook was originally intended for writing technical documents related to computer hardware and software but it can be used for any other sort of documentation.
- ▷ DocBook content is presentation-neutral and can be published in a variety of formats, including HTML, HTML5, EPUB, PDF, man pages and HTML Help, without requiring users to make any changes to the source.
- ▷ DocBook began in 1991 as a joint project of HAL Computer Systems and O'Reilly & Associates. Since 1998 it is maintained by a Technical Committee at OASIS.

DocBook Elements

- ▷ DocBook provides about 400 content markup tags
- ▷ Structural Elements: specify broad characteristics of their contents, e.g. book, part, article, chapter, appendix, dedication
- ▷ Block-level Elements: specify structured blocks of text (usually starting and ending with new "lines"). e.g. paragraphs, lists, definitions, etc. They usually have a fixed content model; some can contain text.
- ▷ Inline-level Elements: wrap text within a block-level element (usually without

breaking "lines"), e.g. for emphasis, hyperlinks, definienda,. They typically cause the document processor to apply some kind of distinct typographical treatment to the enclosed text.

DocBook Example

▷ A "Hello World" document in DocBook

```
<?xml version="1.0" encoding="UTF-8"?>
<book xml:id="simple_book" xmlns="http://docbook.org/ns/docbook" version="5.0">
  <title>Very simple book</title>
  <chapter xml:id="chapter_1">
    <title>Chapter 1</title>
    <para>Hello world!</para>
    <para>
      I hope that your day is proceeding
      <emphasis>splendidly</emphasis>!
    </para>
  </chapter>
  <chapter xml:id="chapter_2">
    <title>Chapter 2</title>
    <para>Hello again, world!</para>
  </chapter>
</book>
```


10.2 Topic-Oriented Documentation with DITA

DITA the "Darwin Information Typing Architecture"

▷ **Definition 10.2.1** DITA is a topic-oriented content markup language for technical documentation based on XML. It supports a topic-oriented documentation style.

▷ **Definition 10.2.2** The basic unit of information in DITA is a topic, i.e. a discrete piece of content that is about a specific subject, has an identifiable purpose, and can stand alone (does not need to be presented in context for the end-user to make sense of the content).

▷ Topics can be reused in any context; DITA makes use of this.

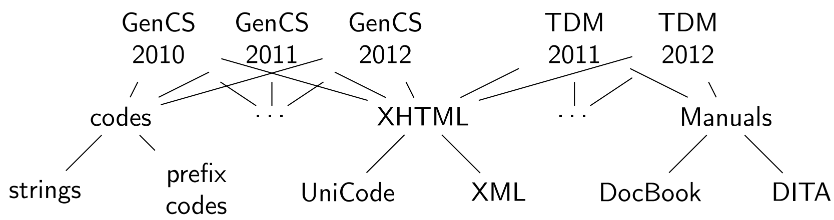
▷ **Definition 10.2.3** DITA combines topics into documents via DITA map s.

▷ Consequence: A DITA topic (and DITA map) can be referenced in multiple DITA maps.

▷ Extension: Conditional text allows filtering or styling content based on attributes for audience, platform, product, and other properties. (the DITA processor filters text)

Using DITA Maps for Reuse

- ▷ Idea: Concepts can be reused in more than one DITA map
- ▷ **Example 10.2.4** For instance a module on HTML/XML in the courses "General Computer Science" and "Text and Digital Media".



Courses given in different years share most of their content (but not all)

A DITA Concept File

- ▷ **Definition 10.2.5** A DITA concept is a special DITA topic that describes an abstract idea or a named unit of knowledge.
- ▷ **Example 10.2.6** A concept for "academic conference" (note the conditional text)

```
<concept id="A.dita">
  <title>Academic Conference</title>
  <conbody>
    <p audience="students">
     An <term>academic conference</term> is a gathering of scientists
     who discuss <term>scientific papers</term>.
    </p>
    <p audience="professors">
      An <term>academic conference</term> is a pretense to travel to
      nice locations on university money and drink loads of beer.
    </p>
    <para conref="#topic/p2"/>
  </conbody>
  <related-links>
    <linkpool type="concept">
      <link audience="students" href="http://easychair.org"/>
      <link audience="professors" href="http://acapulco.mx"/>
    </linkpool>
  </related-links>
</concept>
```

We can generate two versions from this content markup format. For instance, with the following DITA value specification:

```
<!-- this file specifies the actions for students -->
<val>
  <prop action="exclude" att="audience" val="professors"/>
  <prop action="include" att="audience" val="students"/>
</val>
```

A DITA Task File

▷ **Definition 10.2.7** A DITA task is a special DITA topic that describes a process.

▷ **Example 10.2.8** DITA task markup for assignment 8 of the TDM course

```
<task id="TDMassignment8">
  <title>Assignment 8: Reviewing Papers</title>
  <taskbody>
    <prereq>You have to be a registered TDM student.</prereq>
   <steps>
      <step>
        <cmd>accept the PC invitation, log into easychair</cmd>
        <info>You should have been given the information in the
                  invitation e-mail</info>
      </step>
      <step>
        <cmd>indicate your conflicts of interest</cmd>
        <info>you have a conflict with anybody you have a relationship that
          would keep you from being objective (yourself, your family members,
          loved/hated ones, group members,... be honorable)
        </info>
        <stepresult>
          <p>The system records a list of conflicted paper and will not show
                 you anything about them.</p>
        </stepresult>
      </step>
    </steps>
  </taskbody>
</task>
```

  JACOBS UNIVERSITY

A DITA Map File

▷ **Definition 10.2.9** A DITA map combines DITA topics and maps into a document by transclusion.

▷ **Example 10.2.10**

```
<map>
  <title>Life as an Academic</title>
  <topicmeta>...</topicmeta>
  <topicref href="introduction.dita" collection-type="sequence">
    <topicref href="conference.dita"/>
    <topicref href="TDMassignment8.dita"/>
  </topicref>
  <reltable>
    <relcell>conference.dita</relcell>
    <relcell>TDMassignment8.dita</relcell>
  </reltable>
</map>
```

  JACOBS UNIVERSITY

Bibliography

[BCHL09] Bert Bos, Tantek Celik, Ian Hickson, and Høakon Wium Lie. Cascading style sheets level 2 revision 1 (CSS 2.1) specification. W3C Candidate Recommendation, World Wide Web Consortium (W3C), 2009.

[BLFM05] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform resource identifier (URI): Generic syntax. RFC 3986, Internet Engineering Task Force (IETF), 2005.

[CS14] Scott Chacon and Ben Straub. *Pro Git*. APress, 2nd edition edition, 2014.

[CSFP04] Ben Collins-Sussman, Brian W. Fitzpatrick, and Michael Pilato. *Version Control With Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[Dri10] Vincent Driessen. A successful git branching model. online at http://nvie.com/posts/a-successful-git-branching-model/, 2010.

[DW] Marisa DeMeglio and Daniel Weck. Epub media overlays 3.0.1. http://www.idpf.org/epub/301/spec/epub-mediaoverlays.html.

[ECM09] ECMAScript language specification, December 2009. 5^{th} Edition.

[EV] Epub validator (beta). http://validator.idpf.org/.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force (IETF), 1999.

[GMCE] Markus Gylling, William McCoy, Dave Cramer, and Elika J. Etemad. Epub content documents 3.0.1. http://www.idpf.org/epub/301/spec/epub-contentdocs.html.

[GMG] Markus Gylling, William McCoy, and Matt Garrish. Epub publications 3.0.1. http://www.idpf.org/epub/301/spec/epub-publications.html.

[GO] Project gutenberg. Project page at https://www.gutenberg.org. accessed 1. 3. 2016.

[HBF+14] Ian Hickson, Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, and Silvia Pfeiffer. HTML5. W3C Recommentation, World Wide Web Consortium (W3C), 2014.

[HL11] Martin Hilbert and Priscila López. The world's technological capacity to store, communicate, and compute information. *Science*, 331, feb 2011.

[IDP] International digital publishing forum (idpf). http://www.idpf.org.

[Kay08] Michael Kay. Saxonica: XSLT and XQuery processing. http://www.saxonica.com, 2008.

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, August 2006.

[Koh08] Michael Kohlhase. Using LaTeX as a semantic markup format. *Mathematics in Computer Science*, 2(2):279–304, 2008.

[Koh16] Michael Kohlhase. sTeX: Semantic markup in TeX/LaTeX. Technical report, Comprehensive TeX Archive Network (CTAN), 2016.

[PG] James Pritchett and Markus Gylling. Epub open container format (ocf) 3.0.1. http://www.idpf.org/epub/301/spec/epub-ocf.html.

[RE] `re` – regular expression operations. online manual at https://docs.python.org/2/library/re.html.

[RHJ98] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.0 Specification. W3C Recommendation REC-html40, World Wide Web Consortium (W3C), April 1998.

[Vei] Daniel Veillard. The xslt c library for gnome; the xsltproc tool. System Home page at http://xmlsoft.org/XSLT/xsltproc2.html.

[Vol11] Victor Volkman. Classic parsing with flex and bison. http://www.codeguru.com/csharp/.net/net_general/patterns/article.php/c12805__2/Classic-Parsing-with-Flex-and-Bison.htm, 2011. visited Feb 2011.

[XAM] apache friends - xampp. http://www.apachefriends.org/en/xampp.html.

[XML] Extensible Markup Language (XML) 1.0 (Fourth Edition). Web site at http://www.w3.org/TR/REC-xml/.

Index