Einführung UNIX (WS 2016/17)

Martin Held

FB Computerwissenschaften
Universität Salzburg
A-5020 Salzburg, Austria
held@cosy.sbg.ac.at

9. Jänner 2017

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Personalia

LVA-Leiter:	G. Eder, M. Held.
Email-Adresse:	`geder@cosy.sbg.ac.at`, `held@cosy.sbg.ac.at`.
Büro (G. Eder):	FB Computerwissenschaften, Zi. 0.28, Jakob-Haringer Str. 2, Salzburg.
Büro (M. Held):	FB Computerwissenschaften, Zi. 1.20, Jakob-Haringer Str. 2, Salzburg.
Telefonnummern:	8044-6304 (M. Held), 8044-6327 (G. Eder), 8044-6328 (Sekr.).
Basis-URL:	`http://www.cosy.sbg.ac.at/~held`.
WWW-Homepage:	Basis-URL`/held.html`.
LVA-Homepage:	Basis-URL`/teaching/unix/unix.html`.
Allg. Information:	Basis-URL`/for_students.html`.

Formalia

Abhaltezeit der LVA: Der Vorlesungsteil dieser Lehrveranstaltung wird jeweils sine tempore im T01 wie folgt für alle Parallelgruppen gemeinsam abgehalten:

am:	MO 3.10., 10.10., 17.10., 24.10., 7.11. und MO 21.11.2016
um:	12^{40}–15^{45}.

Der Übungsteil dieser Lehrveranstaltung wird in etlichen Parallelgruppen im Zeitraum Oktober bis Dezember 2016 abgehalten. (Siehe LVA-Homepage.)

Achtung: Für den Übungsteil ist eine separate Anmeldung erforderlich!

Lehrziele und Lehrinhalte

Die Lehrveranstaltung VP „Einführung UNIX“ ist ein Service von uns an Sie. Ziel dieser Lehrveranstaltung ist es, Sie *alle* mit Unix-Computern so weit vertraut zu machen, dass Sie *einfache Arbeiten* auf diesen Computern *effizient und sicher* selbst ausführen können, ohne bereits an den Eingangshürden zu scheitern und ohne ununterbrochen in den Manuals blättern zu müssen — auch wenn Sie bisher wenig oder noch gar keinen Umgang mit (Unix-)Computern hatten!

Insbesondere sollte die Kenntnis der Unix-Dienste und Unix-Werkzeuge Sie in die Lage versetzen, im Rahmen des weiteren Studiums benötigte „Produkte“ (zum Beispiel Programme) leichter herzustellen, dabei Arbeit und Zeit zu sparen, und eine ordnungsgemäße Abgabe zu schaffen.

Es ist definitiv nicht das Ziel, in Windeseile wichtige Themen aus „Betriebssysteme“, „Rechnerarchitekturen“ oder verwandten Lehrveranstaltungen detailliert zu besprechen. Viele Themen werden daher zwangsweise und bewusst nur sehr ungenau oder stark vereinfacht diskutiert werden. Eine detailliertere Behandlung wird im Rahmen Ihres Studiums in nachfolgenden Lehrveranstaltungen geboten werden.

Beurteilungsrichtlinien

Die Beurteilung findet mittels Mitarbeit in der Lehrveranstaltung sowie zweier Klausuren am 13.1.2017 (14^{00}–16^{00}) und 10.3.2017 (14^{00}–16^{00}) statt, von denen genau eine Klausur zu absolvieren ist. Bitte beachten Sie, dass

- die Teilnahme an der Abschlussklausur nur für jene Studierenden möglich ist, welche für den Übungsteil dieser Lehrveranstaltung angemeldet waren und daran auch regelmäßig teilgenommen haben;
- eine Teilnahme an einer Klausur nur nach erfolgter Anmeldung zu dieser Klausur (im PLUSonline) möglich ist;
- es definitiv keinen dritten oder weiteren Klausurtermin geben wird;
- ein Nichtantreten bei beiden Klausurterminen automatisch eine Beurteilung mit "NGD5" nach sich zieht;
- ein Antreten zum zweiten Termin nicht möglich ist, wenn bereits die erste Abschlussklausur absolviert wurde, egal mit welchem Erfolg;
- zur Klausur Schreibmaterial sowie die Unicard oder zumindest ein (aktueller) Lichtbildausweis mitzunehmen ist;
- Bücher, Notizen oder andere Unterlagen nicht zugelassen sind; auch keine Wörterbücher.

Prüfungsimmanente Lehrveranstaltung

Achtung: Diese Lehrveranstaltung ist *prüfungsimmanent*! Dies bedeutet aufgrund des geltenden Studienrechts sowie entsprechenden Weisungen des Rektorats,

- dass Anwesenheitspflicht während der gesamten Dauer der Lehrveranstaltung herrscht;
- Leistungsüberprüfungen jederzeit und ohne Vorankündigung möglich sind;
- dass auch bei einem Abbruch dieser Lehrveranstaltung eine Beurteilung an Hand der (bis zum Abbruch) erbrachten Leistung erfolgt, sofern der Abbruch samt Abmeldung nicht vor der dritten UV-Einheit erfolgt; in der Praxis bedeutet dies bei einem Abbruch fast immer eine Beurteilung mit NGD5;
- dass nach erfolgter Beurteilung eine Verbesserung der erhaltenen Note nur durch neuerliches Absolvieren der gesamten Lehrveranstaltung in einem späteren Semester erfolgen kann!

Lehrbehelf — Worte der Warnung

Die von uns in dieser Lehrverstaltung verwendeten Dias stehen Ihnen als PDF-Datei auf der Home Page dieser Lehrveranstaltung als Lehrbehelf zur Verfügung. Eine Kopiervorlage mit allen Dias können Sie auch im Sekretariat des Fachbereich Computerwissenschaften (Jakob-Haringer Str. 2, Zimmer 1.18a) zwecks Anfertigung eigener Kopien ausborgen.

Wir behalten uns allerdings vor, in der Lehrveranstaltung auch über Themen zu sprechen (und Sie dann auch darüber zu prüfen), welche nicht notwendigerweise in den Dias aufscheinen! Prüfungsstoff der Lehrveranstaltung ist alles, was in der Lehrveranstaltung besprochen wird — egal ob es in den PDF-Dias steht. Keinesfalls sollte die Bereitstellung der PDF-Dias als Lehrbehelf für Sie dazu führen, dass Sie diese Lehrveranstaltung nicht mehr regelmäßig besuchen.

Acknowledgments

I am happy to acknowledge that I benefited from material published by colleagues on diverse topics presented in this lecture. In particular, sections of the text have been taken verbatim from the online Unix/Linux man-pages. While some of the material used for this lecture was originally presented in traditional-style publications (such as textbooks), many more items have their roots in non-standard publication outlets (such as online documentations, tutorials, FAQ's, man-pages, or user manuals). Many of the slides in the section on Communication and Data Transfer were created by Regina Bernhaupt. Recent revisions of the slides were carried out by Tobias Berka, Günther Eder, Dominik Engel, Stefan Huber, Stefan Lukesch and Andreas Maier.

Salzburg, July 2016 Martin Held

Legal Fine Print and Disclaimer

Recommended Textbooks

- A. Robbins.
  UNIX in a Nutshell. 4th edition.
  O'Reilly, 2005. ISBN 9780596100292.
- S. Powers, J. Peek, T. O'Reilly, M. Loukides.
  Unix Power Tools. 3rd edition.
  O'Reilly, 2003. ISBN 9780596003302.
- C. Easttom, B. Hoff
  Moving From Windows To Linux. 2nd edition.
  Charles River Media, 2004. ISBN 9781584504429.

Table of Content

What is Unix?

- Unix, like Windows and Mac OS X, is an operating system, or OS for short.
- An OS is a software system that a computer runs when you first turn it on.
- An OS is not an application program like Microsoft Word and is not a set of programs like Microsoft Office.
- Every personal computer has to have an OS or it will not work in the way you are used to see it work.
- Basically, an OS coordinates all of a computer's components into a single, integrated unit.
- Examples of proprietary Unix systems are Mac OS X (by Apple), HP-UX (by Hewlett-Packard, and Solaris (by Sun Microsystems, now Oracle).

Spelling

The standard spelling is “Unix”, with the term “UNIX” being reserved for the trademark. (The present owner of the trademark is The Open Group.) In its early days the spelling “UNIX” was also widely used.

What is Linux?

- Linux is a Unix clone (OS) created by Linus Torvalds when he was a student at the University of Helsinki.
- To be precise, Linux is the so-called kernel of an OS.
- The job of the kernel is manifold: providing interfaces to hardware, multi-tasking programs, managing memory, offering file system access, etc.
- All other applications and utilities needed for a full-fledged Unix OS are programs re-compiled to run on Linux.
- Typical examples for such application programs are file editors, mail readers, etc.
- Different groups or organizations package all those utilities into different distributions: see, e.g. (in alphabetical order) Arch, Caldera, Debian, Fedora, Gentoo, Mandriva, openSUSE, Red Hat, and Ubuntu.
- The Linux kernel is distributed under the GNU General Public License (GPL).

Excerpt from the GNU GPL

"When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.
To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.
For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights."

- Many different "GPL-like" licenses are used in practice — make sure to read the full license terms in order to catch the (sometimes subtle) differences!

Typical Flavors of Software Licenses

Public Domain: Public domain software is software that is not copyrighted. It is software that you are allowed to copy, modify, redistribute, include into commercial products in any form, and even make money by doing so.

Free: Free software, in the sense of ‘freedom’ to modify and redistribute, is the preferred term for software governed by a copyright license that follows the spirits of the GPL.

Open Source: Open Source is a catch phrase that is ambiguous. It may refer to “free” software in the spirits of the GPL, but it may also refer to code which can be incorporated into commercial packages (for free) where later versions will be unavailable in source form. However, the differences are slim: nearly all free software is open source, and nearly all open source software is free.

Shareware: Shareware refers to software that is encouraged to be redistributed, but which requests a small royalty if you end up using it. Shareware regulations come in a variety of flavors; sometimes, license fees are only required for non-academic or site-wide use.

Proprietary: One is required to purchase the software (or a license key) prior to being allowed to use it.

Terminology

- *Multi-tasking* refers to the ability of an OS to execute several processes ("tasks") pseudo-simultaneously on one or more CPUs.
- In *preemptive multi-tasking*, the OS decides when a process is interrupted from being executed by a CPU and another process is re-assigned to a CPU. In general, the switching times are so short that it gives the appearance of executing all processes in parallel.
  The short time sections in which a process is executed are called a "time slice". The length of the time slices depends on the priority or the usage of resources like memory or I/O.
- In *cooperative multi-tasking*, a process currently controlling the CPU needs to offer control to other processes in order to allow them to be executed. It can hog the CPU, though.
- *Multi-user* refers to the ability of an OS to administer two or more local or remote users, and to protect them (and their processes) from each other.
- *Multi-processor* refers to the ability of an OS to administer two or more CPUs.

Pros of Unix/Linux

- Unix/Linux is a true multi-tasking, multi-user and multi-processor OS.
- Unix/Linux almost never freezes under normal use. (Normal use means anything but changing the OS itself.)
- Unix/Linux machines are known to run for months without a reboot. (The main issue in Unix/Linux stability is the hardware it runs on!)
- Unix/Linux scales well across a wide range of hardware resources
- Unix/Linux can be fully customized to fit specific needs.
- Linux is very portable, and runs on virtually any computer hardware.
- Tons of Internet documents provide help and advice.
- Linux is entirely cost- and license-free.

Cons of Unix/Linux

- Windows and Mac OS X have a GUI (graphical user interface). Traditionally, Unix/Linux has sported a CLI, a command line interface. However, nowadays also Linux distributions come with a variety of GUI systems.
- Many features of Unix/Linux are of a technical nature, and require patience to learn, and experience to fully understand.
- To some extent, Linux still requires that you learn to perform administration tasks like adding new users and installing software. (Recent Linux distributions, like Ubuntu Linux, have made maintenance relatively easy, though.)
- There is no central support for Linux unless you buy it through companies tailored to provide it.
- Finally, you use Linux at your own risk. There is nobody to sue!

Early History of Unix

- In 1965, Bell Labs was working with General Electrics on an OS called “Multics”, but the project did not get anywhere.
- In 1970, Ken Thompson of Bell Labs implemented a bare-bones OS for a PDP-7, based on his ideas and ideas of Dennis Ritchie.
- Brian Kernighan, also with Bell Labs, suggested to call the new OS “Unix”, as a pun on Multics.
- In 1973, after Dennis Ritchie had developed the C programming language, Unix was rewritten and implemented almost entirely in C.
- When AT&T, which owned Bell Labs, was forbidden from competing in the computing market, it licensed Unix very cheaply to several universities in the late 70’s.

History of Linux

- In 1984, Richard Stallman left MIT and founded GNU. His mission has been to work on “free” software.
- Linus Torvalds, an undergraduate student at Helsinki, was dissatisfied with “Minix”, a Unix-like OS written by Andrew Tanenbaum.
- In 1991, Linus Torvalds had a kernel but no programs of his own, Richard Stallman and GNU had programs but no working kernel.
- By combining the necessary programs provided by GNU and a kernel, Linux was born.
- In subsequent years, people from around the world joined in the work on Linux.
- As of today, nearly 1.000 developers are involved in a single kernel release.
- Linux was officially in Beta testing until the release of version 1.0 on 14th March, 1994.

Login Procedure

- Turn on your Unix box. You will then see a so-called “login prompt”, which will state (perhaps in addition to other things) the name of the computer and the word `login:`.
- Most likely, the login prompt will be neatly wrapped inside a graphical login and a graphical session will be started after login.
- Now you should type your *user name*, and then press “return”.
- A “password prompt” will appear after which you should type your *password* (“pwd”), and then press the “return” key again.
- Depending on the login system that is used your password will not be displayed in plain text while typing. Usually the password is obfuscated by replacing each character with $\star$ or other symbols, or not printed at all (e.g., for a terminal login).
- If the system re-displays the login prompt then the uid/pwd combination that you entered was unknown to the systems. E.g., you might have mistyped your password. Try again.
- The login procedure is necessary in order to tell the system who is using it. Recall that Unix is a multi-user system!

Changing Your Password

- After your first-time login, your very first task is to change the password assigned to you by the creators of your Unix *account*.
- The command or procedure to be used for changing a password is site-dependent.
- To change the password for your Unix account here at the Department of Computer Sciences, use the command `passwd`.
- You will have to enter your current password. You will then be asked for a new password, and will have to confirm the new password.
- If the password chosen is accepted by the system then it will take effect within a few moments.
- Note that this will change the password for your account and all Unix machines of the Department (and not just the password for a single computer terminal). So no matter which computer in our network you log on to, use the new password.

A Note on Passwords

- The password is the authentication procedure used by the computer to verify that that someone logging in with your uid is really you.

Warning

If someone else obtains your password, they can use and misuse the computer, including manipulating or destroying your data, or even performing illegal activities in your name.

- Unfortunately, in such cases, it is difficult to find out who the real culprit was, and you might be subject to prosecution.
- Also, any computer system, no matter how secure it is from network or dial-up attack, Trojan horse programs, and so on, can be fully exploited by an intruder if he or she can gain access via a poorly chosen password.
- Thus, by choosing a poor password, you do not only put at risk the security of your own data but create a security hazard for the entire local network.

Poorly Chosen Passwords

Do not use

- your login name in any form (as-is, reversed, capitalized, doubled, etc.).
- your first, middle, or last name in any form.
- your spouse's or child's name.
- other information easily obtained about you, like license plate numbers, telephone numbers, social security numbers, etc.
- a word contained in dictionaries, spelling lists, or other lists of words, even when using "obscure" languages or words.
- names of science fiction characters, chemical names, astronomical objects, etc. If you can think of it, so can someone else!
- a password shorter than twelve characters.

Recommendations Concerning Passwords

- Use a password with *at least twelve characters* in total, and at least two non-alphabetic characters (digits and/or punctuation characters).
- Mix lower-case and upper-case characters.
- Keep in mind that German and English keyboards have different layouts and provide different keys.
- If you use different-language keyboards then it may make sense to restrict your password to characters contained in the common intersection of the characters provided by the corresponding keyboards.
- Good passwords are based upon non-dictionary words.
- A good way to pick a password is to use the first letter of a phrase that you can easily remember. E.g., “i love Penguin Power” would translate to “ilPP”.
- Another (possibly weaker) method is to intentionally use misspelled words, like “pssawodr”. (Note: transposing only two characters is not enough!!)

Important Advice

In any case, do not write down your password, and do not hand it to anybody else!

Getting Started with Unix

- Unix, and hence also Linux, is traditionally a command-line driven OS. That is, most applications are run from a so-called “prompt”.
- Be warned that Unix is a powerful OS that might turn out to be potentially dangerous for the uninitiated: Always check your command lines before hitting “return”!!
- Unix comes with extensive (online) documentation about pretty much everything you could want to know.
- The first command you should be aware of is `man`: It lets you display online manual pages.
- In case of doubt: RTFM — “read the fabulous manuals”! (Yes, indeed!!)
- In particular, please make sure to consult the department’s local rtfm-server https://rtfm.cosy.sbg.ac.at/doku.php before bothering anyone else with questions about the set-up of the departmental computer network that would quickly be answered by studying the documentation available.

Unix Text Terminal

- A text terminal (text console) is a computer interface for entry and display of *text*. It is the fundamental concept of interaction with a Unix system.
- Non-graphical programs are usually bound to a text terminal to receive (keyboard) input and to display program output and error messages.
- So-called “virtual” terminals are provided on Unix systems that utilize a graphical user interface (GUI) or graphical desktop environment. Such a terminal is a graphical application (usually a single window) that mimics a text terminal and is used to execute non-graphical programs.
- In a GUI the virtual terminal can commonly be found in menus titled “Accessories” or “System Tools”. (Look for `xterm`.)
- Unix commands discussed in this course are to be entered in a (virtual) terminal except when explicitly stated otherwise.
- Note that any Unix command is case-sensitive: `cat` and `Cat` are different commands! (In practice, Unix rarely makes use of different cases.)

Unix Shell

- The first thing you will see in a Unix text terminal is the *prompt*.
- A prompt may consist of as little as one character, e.g., "`>`" or "`:`", but it might also display more information like "`79 held@ursus->`".
- In any case, as its name suggests, it is prompting you to enter a command. Note that every Unix command has to be completed by pressing the "return" key.
- The program that displays the prompt is called a *shell*.
- The shell's job is to interpret the commands and run the programs you request.
- Each user is allocated his/her own shell at login.
- Since Unix was designed to be a multi-tasking and multi-user OS, it can run multiple shells.
- Shells can also be programmed in their own language, and programs written in that language are called *shell scripts*.

Unix Shell

- There are two major types of shells in Unix: *Bourne shells* and *C shells*.
- Bourne shells are named after their inventor, Steven Bourne, who wrote the original Unix shell `sh`.
- C shells, `csh`, which were originally implemented by Bill Joy, are also very common.
- Traditionally, Bourne shells have been used for shell scripts and compatibility with the original `sh`, while C shells (`csh`, `tcsh`) have dominated interactive use.
- Most Linux systems come with a Bourne shell called `bash` (for "Bourne Again Shell") as default shell.
- Bash shells combine all the standard programming features of Bourne shells with many interactive features commonly found in C shells.
- In recent years, the functional differences between `tcsh` and `bash` have blurred, and it is mostly a matter of taste which shell is used.
- Your Unix account will be set up with `bash` as default shell.

Files

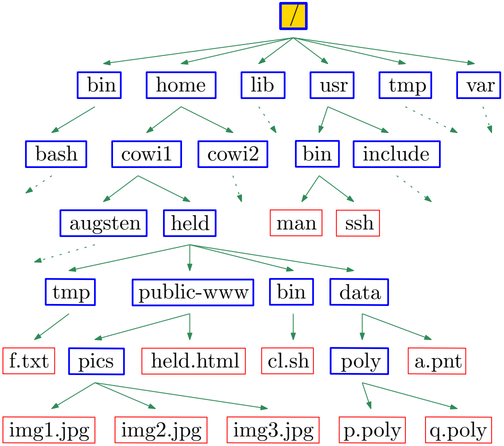
- Common to every computer system invented is the *file*: it holds a single contiguous block of data.
- A filename on a Unix machine can be up to 256 characters long.
- Although Unix filenames can contain almost any character, practical standards dictate that only the following characters are to be used in filenames:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -
```

- The use of Germanic "umlauts" in file names may cause troubles for certain applications.
- You should never use other punctuation characters, brackets or control characters in filenames. Also, never use the space or tab character in a filename.
- Needless to say, filenames should be chosen such that they bear some connotation with the data they contain.

Directories

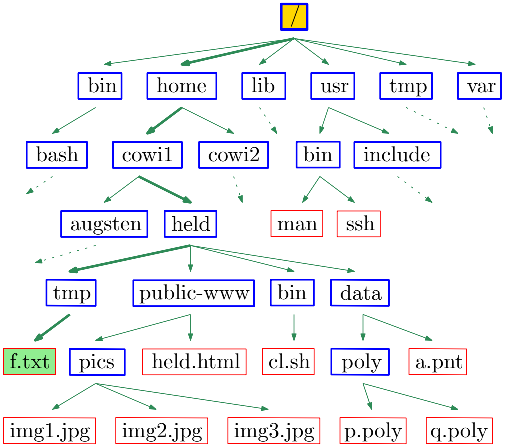
- Files are organized within *directories*.



- At any time, a user is said to be "in" one of the directories.
- The first (top-most) directory is the *root directory*, denoted by the symbol /.
- Every file and every directory lives in / or can trace its location, or *path*, through a hierarchy of directories to a directory that lives in /.

Directories: Absolute Pathnames

- The location of any file can be specified by its *absolute pathname*, or by a *relative pathname*.

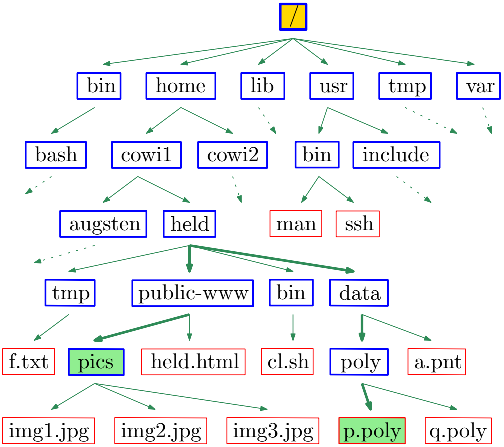


- Absolute paths always start with /.
- The names of directories in a path are separated by /.
- E.g., absolute path to `f.txt`: `/home/cowi1/held/tmp/f.txt`

Note
The first / in an absolute path is the “name” of the root directory, while all other / symbols are meta symbols used to limit the names of directories.

Directories: Absolute Pathnames

- Relative paths are relative to the directory you are currently in.



- Hence, a relative path depends on your current directory and, likely, will not make sense when your current directory has changed.
- The parent directory of the directory you are currently in is specified by .. (that is, by two dots).
- E.g., relative path from the directory `pics` to the file `p.poly`: `../../data/poly/p.poly`

Listing of Basic Unix Commands

Directory functions:

- `pwd`
  short for “print working directory”; this shows you the directory you are currently in.
- `cd <dir>`
  short for “change directory”; this allows you to “move into the directory” named `<dir>`.
- `cd ..`
  move into the parent directory of your current working directory.
- `cd`
  move into your home directory.
- `ls`
  lists files in the current working directory.
- `ls <dir>`
  lists files in the directory named `<dir>`.

Listing of Basic Unix Commands

Listing the contents of files:

- `more <file>`
  for viewing files; the file `<file>` will be displayed in sections that fit into your screen/window.
- `less <file>`
  similar to `more`, but with more functionality.
- `cat <file1> ...`
  short for "concatenate"; this is the simplest way of viewing one or more files.

Listing of Basic Unix Commands

Creating and moving files and directories:

- `cp <file1> <file2>`
  short for "copy"; copies files and directories.
- `mv <file1> <file2>`
  short for "move"; moves files and directories.
- `mkdir <dir>`
  short for "make directory"; creates a new directory.
- `rm <file>`
  short for "remove"; deletes a file – use with caution!
- `rmdir <dir>`
  short for "remove directory"; deletes an empty directory – use with caution!

A Word of Caution

- To prevent accidents, `rmdir` will refuse to remove directories that contain files and/or other directories.
- In any case, you should be very careful with `cp`, `mv`, and `rm`!

Warning

1. Note that neither `cp` nor `mv` checks whether a file already exists, and will remove any old file in its way.
2. Furthermore, note that Unix does not provide any system-inherent means for recovering files that were accidentally removed.

- However, there is a way to make these commands ask for a confirmation: all three of these commands accept the `-i` option, which makes them query the user before actually performing the operation.

Listing of Basic Unix Commands

Finding files and commands:

- `man <app>`
  display the online reference manual for the application.
- `which <app>`
  tells you where to find the binary of an application.
- `locate <pattern>`
  lists files in databases that match a pattern.
- `whatis <app>`
  displays manual page descriptions.
- `apropos <app>`
  searches the manual page names and descriptions.

Working with Unix

X Window System

- The X Window System is a *network-transparent window system* for producing graphical user interfaces (GUI's) in a networked environment.
- It is maintained by the X.Org Foundation, which was founded on 22-January-2004.
- The X Window System standard was originally developed at MIT's CS Laboratory, with development work started in 1984. Since its 11th release, in 1987, it has been considered stable and, thus, got to be known as "X11".
- All rights to it were assigned to the X Consortium, which was the predecessor of the X.Org Foundation, on 01 January 1994.
- It is commonly called "X11R7", to denote that the current release is version 11, release 7. (The current version X11R7.7 was released on 06-June-2012.)
- X11 is the industry standard for window systems on bitmap displays linked to Unix/Linux machines.
- Local and network-based computing look and feel the same, and distributed applications can be realized easily.
- Mac OS X does no longer support X11 directly, but X11 server and client libraries for OS X are available from the XQuartz project. (Apple is a contributor to the XQuartz project and seems to support the integration of X11 into OS X.)
- For Windows 8/7/Vista/. . ., Xming has provided X11 Servers.

Client-Server Model

- X11 is based on the *client-server model*.
- An X *client* is an application or program, such as a clock or a calculator.
- The X *server* acts as an intermediary between the client (application) and the I/O devices (e.g., keyboard and display).
- The server takes information from an input device, such as the keyboard or a mouse, and sends it to the client. It also takes information from the client and sends it to the graphics hardware.
- Typically, the server and a client run on the same workstation.
- Clients, however, can be run from other machines via a local network or even via the Internet, involving machines with different operating systems.
- Note that such a "remote display" requires you to enable X11 forwarding.

X Protocol

- The communication between the X server and an X client is carried out via the standardized *X protocol*.
- The X protocol defines a client-server relationship between an application (the X client) and its display (the X server).
- The X protocol hides the peculiarities of the OS and the underlying hardware.
- This masking of architectural and engineering differences simplifies X client development and provides the springboard for the X Window System's high portability.
- The X server is highly portable allowing support for a variety of languages and operating systems.
- X clients also have a high degree of portability.

Window Manager and Desktop Environments

- Unix and X11 allow you to interact with your computer via a variety of different graphical features, and, most notably, via so-called *windows*.
- A *window manager* is essentially the component which controls the appearance of windows and provides the means by which you can interact with them.
- Virtually all windowing systems allow you to move, resize, maximize and also to minimize/iconify windows, i.e., to shrink a window to a small icon when it is not needed.
- Most window managers are regular (albeit complex) X client programs, and a variety of different GUI's has been built for X11.
- Popular window managers include Xfwm (default for Xfce), KWin (default for KDE), Metacity (default for Gnome 2.x), Mutter (default for Gnome 3), IceWM (compatible to Gnome and KDE), Marco (default for MATE), and Compiz (based on OpenGL rather than X!).
- So-called *desktop environments*, such as Gnome, KDE, Xfce, Unity or MATE, aim to provide an even more complete interface to the OS than a standard window manager.
- E.g., desktop environments offer a file manager, task field, toolkit, options for configuring the system, menus for running applications, etc.

Users and Groups

- Every user has a personal *home directory*, typically `/home/<username>` or some variation thereof, in which their own files are stored.
- Other than the super-user (sysadmin), every other user has limited access to files and directories.
- Users are also divided into so-called *groups* — a user may belong to several groups.
- Every file on a system is owned by a particular user and is also associated with one of his groups.
- Use `ls -l` to list the user and group information for files.
- The ownership of files can be changed by using the `chown` command.
- Only the owner of a file (or the super-user) may change the ownership information of that file. For security reasons, sysadmins prefer to disable the `chown` command.

File Protections

- Every file and directory has access flags specifying what kind of access that user and group has to the file.
- This is an extract from a listing of my home directory:

```
drwx------   2 held     inst           512 Oct 10 16:39 Mail/
-rwx------   1 held     inst             7 Oct 10 18:56 up.sh*
drwxr-x---  12 held     compgeo        512 Sep  6 11:04 papers/
drwx------  54 held     inst          1024 Sep 22 17:06 work/
drwx--x--x   9 held     inst          1024 Oct 10 10:17 public/
-rw-------   1 held     inst        149960 Sep  8 11:01 foo.txt
```

- The nine characters after the `d` or `-` are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others.
- Within each set, the three characters indicate the permissions granted. For a file:
  - **r:** Read permission, i.e., the right to gain access to the data stored in the file.
  - **w:** Write permission, i.e., the right to modify the data of the file and to delete the file.
  - **x:** Execute permission, i.e., the right to have the file (program, script) executed by the OS.

File Protections

- Within each set, the three characters indicate the permissions granted. For a directory:
  - **r:** Read permission, i.e., the right to see the contents of the directory, e.g., by issuing the `ls` command. (It does not affect the readability of its files, though!)
  - **w:** Write permission, i.e., the right to create new files/subdirectories in the directory and to delete the directory. (It does not affect the writeability of its already existing files, though!)
  - **x:** Execute permission, i.e., the right to access the directory and its files/subdirectories, e.g., by issuing the `cd` command.
- The `r` and `w` privileges of a directory affect only the directory for which they are speficied but have no effect on its subdirectories.
- A missing `x` privilege of a directory disables all operations that require the name of the directory to be specified in the absolute/relative path to a file/subdirectory, such as access to its files/subdirectories from outside of the directory. Hence, whether or not an `x` privilege is specified for a directory does have an impact on the accessibility of all its subdirectories and the files contained therein!

File Protections

- This is an extract from a listing of my home directory:

```
drwx------   2 held     inst        512 Oct 10 16:39 Mail/
-rwx------   1 held     inst          7 Oct 10 18:56 up.sh*
drwxr-x---  12 held     compgeo     512 Sep  6 11:04 papers/
drwx------  54 held     inst       1024 Sep 22 17:06 work/
drwx--x--x   9 held     inst       1024 Oct 10 10:17 public/
-rw-------   1 held     inst     149960 Sep  8 11:01 foo.txt
```

- The next entries after the file protections show (from left to right)
  - the number of files contained in a directory,
  - the owner and group of the file/directory,
  - the number of bytes occupied by the file or directory entry,
  - the last time that contents of the file/directory were modified, and
  - the name of the file or directory.
- If `ls -F` is used then a trailing slash (`/`) indicates a directory, an asterisk (`*`) indicates an executable file, and an at-sign (`@`) indicates a symbolic link.

Modifying File Protections

- The `chmod` command is used to change the protections of a file or directory: `chmod [-R] [u|g|o|a][+|-|=][r|w|x|s|t] <file>`.
- Check the online documentation for details on the “s” option (setuid-on-execution or setgid-on-execution permission) and the so-called sticky bit “t”.
- The `-R` options means recursive, i.e., to traverse subdirectories recursively.
- Permission bits are often represented in their octal form, where a read permission corresponds to 4, a write permission to 2, and an execute permission to 1.
- For instance, `chmod 0754 <filename>` explicitly sets the permissions for `<filename>` to `rwxr-xr--`. (The leading `0` could be ignored, but is preferred in order to be explicit; it can take on a meaning.)
- The default protection usually is set to `0644`, meaning `rw-r--r--`.
- The shell command `umask` can be used to set the default user file creation mask. E.g., the command `umask 077`, to be specified in an init file, ensures full privacy. (See the documentation for details.)

Unix Commands for Modifying File Protections

Changing ownership and protections:

- `groups`
  shows the groups you belong to.
- `newgrp <group>`
  short for "enter new group"; changes the current group ID. (Only sysadmin can establish a new group, and users are allowed to enter only those groups to which they belong.)
- `chown <usr[:<group>]> <file>`
  short for "change owner"; used to change who owns a file. (Often disabled for security reasons.)
- `chmod <perms> <file>`
  short for "change mode"; used to grant or revoke file access permissions.
- `umask <perms>`
  set the default file protection.

Password File

- If a system like LDAP is not used then, typically, the only place in a Unix system where a user name is registered is in the *password file* `/etc/passwd`.
- This is one line of a password file:

```
held:x:7010:7001:Martin Held:/home/dijkstra/held:/bin/tcsh
```

  It lists (from left to right)
  - the user's login name,
  - the user's encrypted password (or an `x` if a shadow password file is used, see next slide),
  - the user's user identification number, UID,
  - the user's group identification number, GID,
  - the user's full name,
  - the user's home directory, and
  - the shell that is started when the user logs in.

Shadow Password File

- The problem with traditional `passwd` files is that they have to be accessible by a variety of application programs in order to allow them to extract information about the user, such as the user's full name.
- This means that everybody could see the encrypted passwords
- A typical encrypted password is obtained by applying a *one-way hash function* which is known to produce a unique result for every password.
- Trying to guess a password from the hash can only be done by encrypting every possible password and comparing it to the password stored, and is therefore considered computationally expensive but not impossible.
- A so-called *dictionary attack* will simply try tons of words until a match is found.
- In order to avoid making encrypted passwords public, the actual encrypted passwords are stored in a *shadow password file*, `/etc/shadow`, which is only accessible by the system.

LDAP

- LDAP stands for *Lightweight Directory Access Protocol*.
- LDAP offers a directory service that can be used to store and query information about the individuals in an organization.
- LDAP can be used to manage different kinds of information relating to these individuals, like name, contact info, and office location, but also passwords and security keys.
- Utilizing the stored authentication information and an authentication procedure, LDAP can be used to authenticate users when they log on to a computer in the network.
- LDAP is a protocol, not a database. LDAP implementations usually access a special kind of database, optimized for fast reads. (OpenLDAP for example uses the Sleepycat Berkeley DB.)
- LDAP is used at our department for user account management.

Login Procedure

- You use the `login` command at the beginning of each session to identify yourself to the system. `Login` asks for your user name and your password. Where possible, echoing is turned off while you type your password, so it will not appear on the written record of the session.
- If you make several mistakes in the login procedure, all login attempts may be logged in a login log, and you may be silently prevented from further logins for a certain period of time.
- Typical diagnostics include "Login incorrect" (if a user name and password cannot be matched), "Not on system console" (if root login is attempted from an inappropriate source), "No directory!" (if the user's home directory cannot be found; contact your sysadmin), and "No shell" (if the system cannot execute the shell named in the password file; contact your sysadmin).
- At login time, the UID, GID, and home directory are initialized, and the command interpreter (shell) is started.
- You can temporarily become another user by using the *set user command*, `su`.

Getting the Job Done

- Unix is a powerful system for those who know how to use its power.
- We start with examining some of the commands and features that make it easier to “get the job done” in a Unix environment.
- Recall that Unix sports several shells which do not offer an identical functionality.
- Unless noted otherwise, the commands listed in the subsequent slides should work with `tcsh` and `bash`.
- At our Department, `bash` is assigned as the standard login shell to new users. (Currently a user needs to ask our sysadmins to change the LDAP entries if (s)he would like to switch to another default login shell.)

Wildcards

- The `ls` command can produce a lot of output if there is a large number of files in a directory.
- The asterisk “*” and the question mark “?” allow you to define names that match some given pattern. While “*” matches any number of characters, “?” matches exactly one character. E.g.,

  ```
  ls ?oop*.c
  ```

  will list “loop_X11.c” but it will not list “main_loop_X11.c”.
- Please see the documentation on more details for pattern matching.
- Always keep in mind that the asterisk and the question mark have a special meaning for a shell!

Command-Line Editing

- It is important to note that the arrow keys serve an important purpose in a `tcsh` and `bash` shell: They allow you to manipulate shell commands.
- The left and right keys are used for moving backward/forward inside a command that you have typed (but not yet completed by pressing the “return” key).
- The up and down keys are used for moving upward/downward in the command-line history.
- Thus, you can recall a command typed previously, and can then edit it (in a style akin to Emacs) as needed.

Name Completion

- Whenever a filename is needed, you may try typing the first few characters of the filename, and then pressing the “tab” key.
- The shell will then look for a file that starts with those characters and will complete the filename, if it exists and is unique, or beep in order to warn you.
- Similarly for directories.
- If more than one filename matches (and if the shell does not display all possible matches):
  - In a `tcsh`, typing “C-d” (i.e., pressing “control” and “d” together) will cause all possible completions to be displayed if the string that you typed does not allow for a unique completion.
  - In a `bash`, press “tab” a second time to get a list of all possible completions.
- A similar concept works in `tcsh` and `bash` shells for command completions.

History Expansion and Substitution

- Both `tcsh` and `bash` support history expansion, i.e., the inclusion of words from the history list into the input stream.
- This makes it easy to repeat commands, insert the arguments to a previous command into the current input line, or fix errors in previous commands quickly.
- The `history` command lists a (user-defined) number of previously executed commands in chronological order.
- The simplest ways to recycle previous commands are listed below:

`!`*n*	refers to the command numbered *n*,
`!-`*n*	refers to the current command minus *n*,
`!!`	refers to the previous command; identical to `!-1`,
`!<string>`	refers to the most recent command that started with `<string>`.

- Both shells support more complex history substitutions — see the manual pages for details.

Standard I/O and I/O Redirection

- Unix provides three standard input/output channels that allow a program to communicate with you: *standard input* (`stdin`), *standard output* (`stdout`), and *standard error* (`stderr`).
- `Stderr` is virtually always connected to a terminal in order to have a human read the message.
- Data written to `stdout` can be redirected to a file by means of ">" (greater sign).
- For instance, `ls > dir_listing.txt` causes the directory listing to be stored in the file `dir_listing.txt`.
- If the target file does not exist then it is created; if it exists then its data gets overwritten!
- When using ">>", data is appended to a file.
- An interactive program, like a shell, reads commands from `stdin`.
- Similarly to output redirection, instead of reading input from your keyboard, a program can be told to read its input from a file by using the "<" sign.

Pipes and Filters

- *Pipes* can be used to make one program read its input from the output of another program.
- For instance, `ls /usr/bin | more` lists the directory `/usr/bin` in a page-wise mode by feeding the output of `ls` as input into `more`.
- The vertical bar "`|`" indicates the pipe.
- A useful tool in conjunction with pipes are *filters*.
- A filter is a program that reads data from `stdin`, manipulates it, and outputs to `stdout`.
- Typical (albeit simple) filters are `cat`, `sort`, `head` and `tail`.
- For instance, `ls /usr/bin | tail` will only display the last (ten) lines of the directory listing.

Tape Archives

- Several tasks, such as the creation of so-called backups, require the creation and manipulation of duplicates of certain subsets of your files.
- When many files are packed together into one, this packed file is called a *tape archive*. Typically, (tape) archives have the extension `.tar`.
- The following command packs all files of `<directory>` into the file `filename`: `tar -cvf <filename> <directory>`.
- It is common practice to use `<directory>.tar` as `<filename>`.
- It is advisable to watch for any error messages that `tar` reports.
- Also, list the file and check that its size is appropriate for the size of the directory you are archiving.
- A tar file is unpacked by the command `tar -xvf <filename>`.
- Unpacking `<filename>` will re-create your directory with all its files intact.

File Compression

- One of the best utilities for lossless compression is `gzip`, due to its good compression rate and its wide availability on most platforms.
- It is based on the Deflate Algorithm by Phil Katz, which uses Lempel-Ziv 77 and Huffman encoding.
- The default extension for a `gzip`-compressed file is `.gz`.
- The command `gzip [-<rate>] <filename>` causes `<filename>` to be replaced by its compressed version `<filename>.gz`, applying the compression level `<rate>`.
- Maximum compression is requested by 9 while 1 applies least compression.
- The greater the compression, the longer it takes to encode and decode a file.
- Compressed files can be restored to their original form by using `gunzip`.
- The `gunzip` utility can decompress files created by `gzip`, `zip`, and `compress`.
- It recognizes the special extensions `.tgz` and `.taz` as shorthands for `.tar.gz` and `.tar.Z`, respectively.

File Compression

- `bzip2` is another compression tool that generally provides even higher compression ratios than `gzip`, typically at the expense of a (significantly) increased processing time.
- The default extension for `bzip2`-compressed files is `.bz2`.
- The equivalent to `gunzip` is `bunzip2`.
- Both, `gzip` and `bzip2`, can be invoked directly with the `tar`-command by using the flags `z` and `j`, respectively.

Searching for Files

- The command `find` can be used for searching for files.
- Basically, the `find` utility recursively descends the directory hierarchy and, for each path, seeks files that match a Boolean expression.
- For instance, `find . -name '*.c'` will list all files with the extension .c, starting from the current working directory.
- Note that `find` actually traverses a directory hierarchy; it should be expected to be slow when applied to large file systems!

Searching within Files

- The command `grep` does a line-by-line search through a file and prints out those lines that contain a user-specified pattern.
- Its command syntax is `grep [options] <pattern> <filename>`.
- For instance, `grep 'Linux' *.c` searches for all occurrences of the string 'Linux' in the files with extension .c, in the current working directory.
- The option `-i` instructs `grep` to perform a case-insensitive search.
- Be careful when using the characters `$, *, [, ^, |, (, )`, and `\` in the pattern — they are also meaningful to the shell!
- It is safest to enclose the entire pattern in '...', i.e., in single quotes. (Single quotes are located on the same key as # on a German keyboard.)

More Information Commands and Utilities

- The `wc` utility, shorthand for *word count*, lets you count the number of characters, words and lines in a file:
  `wc [-clw] [<filename>]`.
- If `<filename>` is omitted then `wc` operates on `stdin`.
- Piping the output of `ls` to `wc` yields a simple way for counting all files in a directory with, say, the extension .c: `ls -l *.c | wc -l`.
- The `diff` utility, `diff [options] <filename1> <filename2>`, compares the contents of `<filename1>` and `<filename2>` and writes a list of differences to `stdout`.
- No output will be produced if the files are identical.
- It operates on a line-per-line basis, but is reasonably smart and attempts to determine the smallest set of differences (deletions and insertions) that would allow to transform the contents of `<filename1>` into those of `<filename2>`: "Levenshtein distance".
- Details of its use and its diverse options are explained in its manual page.
- The `touch` command updates the time stamps of files.
- It creates an empty file if no file with the name specified as its argument exists.

More Information Commands and Utilities

- The `sort` command, `sort <filename1> [<filename2> ...]`, sorts lines of all the named files together and writes the result to `stdout`.
- The comparisons are based on one or more sort keys extracted from each line of input. The default sort key is the entire input line, relative to alphabetical order.
- The `tr` utility copies from `stdin` to `stdout` with substitution or deletion of selected characters.
- Instead of specific character strings, it can also be applied to classes of characters, such as letters (alpha), numbers (digit), punctuation characters (punct), etc.
- For instance, the following command translates all lower-case characters of `<filename1>` to upper-case and writes the result to `stdout`:
  ```
  tr "[:lower:]" "[:upper:]" < <filename1>
  ```
- The next example outputs all words of `<file1>` in alphabetical order to `<file2>`:
  ```
  tr -cs "[:alpha:]" "[\n*]" < <file1> | sort > <file2>
  ```
- The `tr` utility can be used for a (very simple!) encryption, which dates back to Caesar (100AD–44AD):
  ```
  echo "alea iacta est" | tr 'A-Za-z' 'E-ZA-De-za-d'
  ```
  yields
  ```
  epie megxe iwx.
  ```

Soft and Hard Links

- You may find it convenient to have one file in two different directories at the same time. In order to avoid creating an administrative nightmare by simply copying the file, Unix provides the concept of *links*.
- The command `ln -s <orig_filename> <new_filename>` creates a *soft link*, also known as *symbolic link* (symlink), from `<new_filename>` to `<orig_filename>`.
- The file `<new_filename>` does not contain any actual data but only the path of the file it is linked to.
- A program operating on a link will actually operate on the file (or directory) referenced.
- Links are also applicable to directories, and can span file systems.
- Note: If the object referenced by a symlink is deleted then the symlink constitutes a dangling reference.
- Unix also provides *hard links* — see the manual page.

Introduction to Multi-Tasking

- A *process* is a command/program/shell-script that is being *executed* in memory.
- After a process terminates it is removed from memory.
- Every process is associated with a unique process identification number (PID).
- Every process is owned by somebody; typically the user who initiated the process.
- If a process wants to access a file, its ownership is compared to that of the file, in order to check whether an access is permissible.
- Typically, several dozen processes are running simultaneously at any time when a user is logged in.
- CPU time and other resources are distributed in a so-called *time-sharing mode* among different processes.
- If the main memory is not sufficient to hold all processes, parts of processes will be temporarily moved ("*swapped*") to auxiliary memory like external disks.

Job Control

- Job control refers to a shell's ability to control processes and to put them into the *background* or bring them to the *foreground* again.
- Unless noted otherwise, all job-control commands refer to processes run in the shell in which the command is issued. (To be precise, those commands are shell-builtins.)
- Pressing `^c` ("control" and "c") kills a process, i.e., it terminates it for good.
- Pressing `^z` stops a process, i.e., it suspends its execution.
- The status of a process can be checked using the `jobs` or `ps` command. For instance, the `jobs` command might generate the following listing:
  `[1] + Suspended dummy.sh`.
  The number in brackets is the *job number* of the process.
- A stopped process can be terminated for good by referring to its job number in the `kill` command: `kill %1`.

Job Control

- Typing `fg` causes a stopped process to resume execution in the foreground.
- Typing `bg` causes a stopped process to resume execution in the background.
- A process can also be put into the background right when it is started by adding a trailing ampersand (`&`) to the command: `xemacs foo.c &`.
- Note that a process run in the background is immune to `^`c and `^`z!
- Such a process can be killed by using the `kill` command, or by putting it into the foreground again and then hitting it with an *interrupt* (`^`c).
- The kill command, `kill <PID>`, kills the process whose PID was specified.
- The PID can be listed by means of the `ps` command.
- If `kill`ing a process does not seem to work, use the option `-9` as a last resort: `kill -9 <PID>`.

Obtaining System Statistics

- The `du` command counts the disk space a given directory and all its subdirectories take up. E.g., `du -h --max-depth=1` lists the disk space consumed.
- Note that it depends on the system whether `du` reports *disk usage* in units of 1 byte, 512 bytes, or 1024 bytes — check your manual pages!
- The `df` command, short for *disk filling*, reports for every filesystem (that is mounted) the total amount of disk space, the amount in use, and the amount available.
- The `uptime` command prints the current time, the length of time the system has been up, the current number of users, and the average number of processes in the run queue over the last 1, 5, and 15 minutes.
- The `w` command displays the current users of the system, and what they are doing. Basically, it combines the information given by `uptime` and `who`.
- The `top` command displays the top CPU-consuming processes on the system and periodically updates this information.

A Note on Disk Space

- Please note: Disk space is precious even though a 1TB disk can be bought for less than 100 Euros!
- The key bottleneck is the back-up system!!
- We do not have a fixed disk quota, but we monitor the consumption of disk space.
- Please react if one of our technicians sends an email and requests that you reduce the amount of data stored.
- If you get such a warning:
  - Recall `du -h --max-depth=1` and use it!
  - Check the cache of, e.g., Firefox, Ubuntu Tracker, Java, Google Earth.
  - Check the wastebasket if you use `xfce` (GNOME) or `kde`.
- If you do not react despite of email warnings then access to your account will be suspended!

How to Deal with Problems

1. Check your doing if things do not seem to work the way you expect them to work. This is particularly important for late-night sessions . . .
2. Read manual pages and other sources of documentation to confirm that your intended use of a command/program matches its described use.
3. Think twice before labeling any problem as a “bug” in someone else’s software!
4. If you get an error message, make sure to take enough time to digest the message completely. And, needless to say, act appropriately.
5. Check the available sources of information, such as online documentation and news groups, for whether your problem is known. If it is known then somebody else might have already found (and published) a fix for it.
6. Never post to online forums (like news groups) without prior checking whether your problem is a known problem!
7. As a last resort, seek help by others (sysadmin, news groups, etc.).

How to Deal with Bugs

- If you are absolutely convinced that you have found a genuine bug then you have to make sure that the appropriate information gets to the appropriate place.
- Check the documentation in order to see who wrote the code, and to whom to address bug reports.
- Please be specific when reporting any problem/bug.
- In particular, include as much detail as possible on what you think is wrong, when the problem occurred, on how you used the command/code, in which environment it was used, etc.

Politeness Wins!
In any case, be polite to those to whom you report a problem!! Always remember that Linux is based on volunteer work! (And when dealing with a commercial enterprise it will not help your case either when you piss off their folks . . .)

File Editor

- A file editor is a tool for manually entering or modifying data in files. (Some modern editors are much more powerful, though!)
- Terminal-based editors:

`vi`:	Powerful Unix editor.
`nano`:	Simple text-based editor; it is the license-free sibling of `pico`.

- GUI-based editors with a look&feel similar to MS-Windows:

`gedit`:	Standard editor of the GNOME environment.
`kwrite`:	Standard editor of the KDE environment.

- Two more editors that can be run either in a terminal or GUI-based:

`emacs/xemacs`:	Powerful and highly extendible editor.
`vim/gvim`:	Extended versions of `vi`.

File Editor

- Historically the standard text editor on Unix systems used to be `ed`. It is hardly used any more, except in its streaming version, `sed`.
- Today's standard Unix editor is `vi`.
- It is the only editor which is installed by default on any Unix system.
- Knowing `vi` is imperative for sysadmin work.
- However, new users (and many experienced users ;-) find `vi` unintuitive and tedious to work with.
- Therefore, we provide basic information on XEmacs (and its close relative, Emacs).

XEmacs

- XEmacs is the second defacto-standard editor for Unix systems.
- It is a powerful tool for editing files, sending emails, compiling programs, etc.
- It is started by typing the command `xemacs &` in a shell. (Or `xemacs foo.txt &` to edit the file `foo.txt`.)
- Like most editors, XEmacs uses the concept of a *buffer*: When you open a file for editing, XEmacs copies the contents of the file into a memory region called a buffer and works with this copy.
- XEmacs autosaves your buffer to prevent disaster in case of power loss or a crash. The name of the autosave file is derived from the original file name by enclosing it in a pair of hash marks "#", e.g., `#foo.txt#` for a file `foo.txt`.
- Changes are written back to the actual file only when you save your work. (Typically, the contents of the old file will be preserved in a file whose name is derived by appending a tilde ("~") to the original file name, e.g., `foo.txt~`.)

Basic XEmacs Concepts

- XEmacs is usually in insert mode; that is, text is inserted rather than typed over (overwritten).
- You can toggle between insert and overwrite mode by pressing the “insert” key.
- This is a toggle switch, and pressing this key again gets you back to insert mode.
- XEmacs sports specialized *modes* that offer great support for editing specific types of files, such as C codes, $\LaTeX$ documents, etc. See the documentation for details.
- Virtually every feature of XEmacs can be customized. (More on this later.)

XEmacs Commands

- Experienced users often find it more convenient to enter commands via the keyboard than via manipulating menus.
- Virtually all XEmacs commands involve the “control” key, the “meta” key, or both.
- The standard notation is `C-<key>` for pressing and holding down the “control” key and then pressing the `<key>` key.
- For instance, `C-x C-s` means holding the “control” key down and pressing the “x” key, then releasing the “x” key and pressing the “s” key (while keeping the “control” key pressed).
- Similar for `M-<key>` as a command that uses the “meta” key.
- On keyboards that do not have a “meta” key the “escape” (esc) key often takes the part of the “meta” key. (However, the “escape” key normally is not kept pressed while pressing a second key.)

Sample XEmacs Commands

`C-x C-s`	: Saves the file in the current buffer.
`C-x C-w`	: Prompts for a filename and then saves the file in the current buffer.
`C-x C-f`	: Loads (and creates, if it does not exist) the file that you name.
`C-x C-c`	: Exits XEmacs.
`C-h t`	: Start XEmacs built-in tutorial.
`C-d`	: Deletes the character the cursor is on.
`C-k`	: Deletes the rest of the line.
`C-y`	: Brings back (yanks) what was just deleted.
`C-v`	: Scrolls down one screen at a time.
`M-v`	: Scrolls up one screen at a time.

Sample XEmacs Commands

`C-s` : Search (forward).

`C-f` : Moves the cursor forward, i.e., to the right.

`C-b` : Moves the cursor backward, i.e., to the left.

`C-n` : Moves the cursor down, i.e., to the next line.

`C-p` : Moves the cursor up, i.e., to the previous line.

`C-a` : Moves the cursor to the start of the line.

`C-e` : Moves the cursor to the end of the line.

`M-<` : Moves the cursor to the beginning of the buffer.

`M->` : Moves the cursor to the end of the buffer.

`M-x` : Starts the dialog for more advanced commands; e.g.: `M-x query-replace` as the long version for `M-%`.

XEmacs Window Layout

- The XEmacs window comes with a series of menus at its top, like “File”, “Edit”, “Options”, etc. Those menus provide easy access to some of the more common commands used in XEmacs.
- Below the menus a row of icons for the most common commands is provided.
- The mode line and the command line are at the bottom of the XEmacs window.
- The mode line displays information about the buffer itself, such as the name of the file being edited
- The command line is used to enter commands, path names, and file names from the keyboard instead of using a mouse to click on the menus and icons.
- It is a matter of personal taste whether to use the mouse, menus and cursor keys, or XEmacs commands.

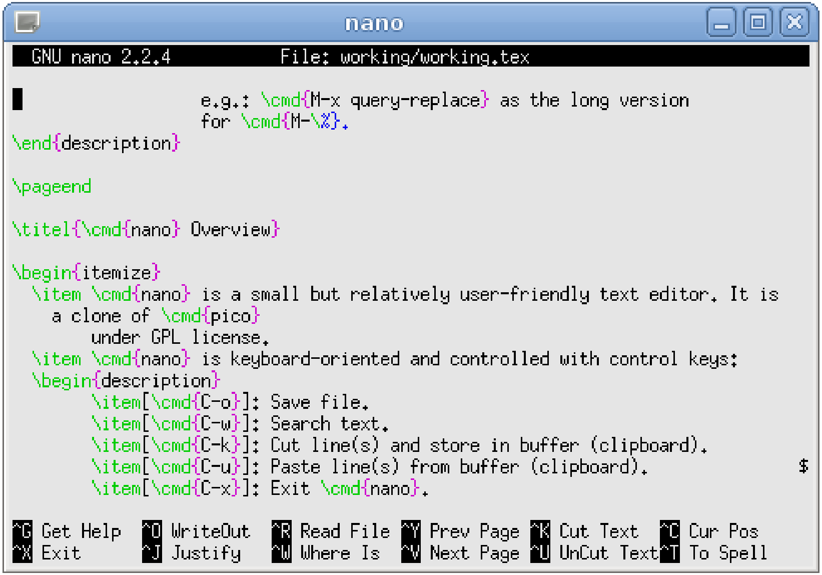
`nano` Overview

- `nano` is a small but relatively user-friendly text editor. It is a clone of `pico` under GPL license.
- `nano` is keyboard-oriented and controlled with control keys:

`C-o`	: Save file.
`C-w`	: Search text.
`C-k`	: Cut line(s) and store in buffer (clipboard).
`C-u`	: Paste line(s) from buffer (clipboard).
`C-x`	: Exit `nano`.

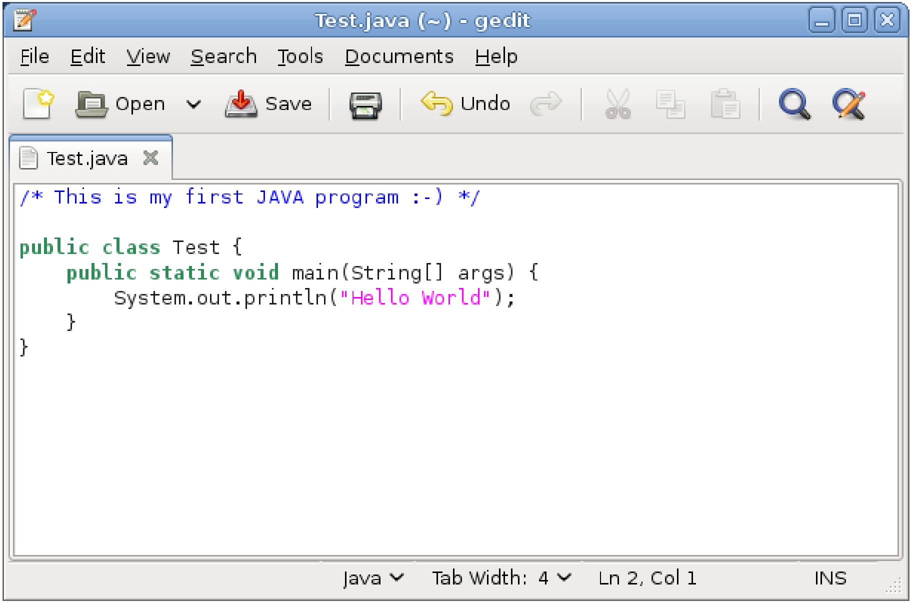
- `nano` displays a two-line "shortcut bar" at the bottom of the screen (so you don't have to remember all commands).

nano



```
 GNU nano 2.2.4              File: working/working.tex

                   e.g.: \cmd{M-x query-replace} as the long version
                   for \cmd{M-\%}.
\end{description}

\pageend

\titel{\cmd{nano} Overview}

\begin{itemize}
  \item \cmd{nano} is a small but relatively user-friendly text editor. It is
    a clone of \cmd{pico}
        under GPL license.
  \item \cmd{nano} is keyboard-oriented and controlled with control keys:
  \begin{description}
        \item[\cmd{C-o}]: Save file.
        \item[\cmd{C-w}]: Search text.
        \item[\cmd{C-k}]: Cut line(s) and store in buffer (clipboard).
        \item[\cmd{C-u}]: Paste line(s) from buffer (clipboard).                $
        \item[\cmd{C-x}]: Exit \cmd{nano}.

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text^T To Spell
```

gedit



```
/* This is my first JAVA program :-) */

public class Test {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Advanced Use of Unix

Customization

Advanced Unix and Shell Programming

Shell Customization

- The design philosophy of Unix was to make it easy for every user to tailor the Unix computing environment to their particular needs.
- This so-called *customization* is mainly done through *configuration* files.
- Configuration files are also known as “init files”, “rc files” (for “run control”), or “dot files” (since the filenames often begin with a dot).
- Note that `ls` lists dot files only if the option `-a` is used.
- The most important configuration files are the ones used by the shell.
- Typically, a shell is configured by setting *environment variables* and defining *aliases*.
- Needless to say, the appropriate configuration commands depend on the shell used. We will cover instructions for both `bash` and `tcsh`.

Shell Start-up and Initialization

- There are essentially two different types of a shell, depending on how it is run: *interactive shells* and *non-interactive* shells.
- An interactive shell is a shell that presents a prompt to you and accepts commands and other input. A typical representative of an interactive shell is a shell run inside of an `xterm`.
- The *login* shell is a special type of interactive shell. It is started at login time, and is the first shell that you will see.
- Non-interactive shells are used for executing commands in batch. A typical representative of a non-interactive shell is a so-called *(shell) script*.

- In the sequel, `~/<filename>` refers to the file `<filename>` in your *home* directory.
- As a meta warning, please be advised that the following slides will barely reveal the most important aspects of customization.

Initialization Files for `bash`

- When `bash` is invoked as a login shell, it first reads and executes commands from the file `/etc/profile`, if this file exists. After reading this file, it looks for `~/.bash_profile`, `~/.bash_login`, and `~/.profile`, in that order, and reads and executes commands from the first one that exists and is readable.
- When a login shell exits, `bash` reads and executes commands from the file `~/.bash_logout`, if it exists.
- When an interactive shell that is not a login shell is started, `bash` reads and executes commands from `~/.bashrc`, if this file exists.
- For non-interactive shells, no init file is sourced by default. See the manual page for details.
- Note that the shell will not automatically re-read any of these files after its start-up. Thus, changes to any of these files will only take effect for the current shell once you `source` the file.

Initialization Files for `tcsh`

- When `tcsh` is invoked as a login shell, it first reads and executes commands from the files `/etc/csh.cshrc` and `/etc/csh.login`, if these files exist. After reading these files, it executes commands from `~/.tcshrc` or, if `~/.tcshrc` does not exist, `~/.cshrc`, then `~/.history`, `~/.login`, and finally `~/.cshdirs`.
- When a login shell exits, `tcsh` reads and executes commands from the files `/etc/csh.logout` and `~/.logout`, if they exist.
- When an interactive shell that is not a login shell is started, `tcsh` reads and executes commands from `/etc/csh.cshrc` and `~/.tcshrc` or `~/.cshrc`, if these files exist.
- For non-interactive shells, no init file is sourced by default.
- Note that the shell will not automatically re-read any of these files after its start-up. Thus, changes to any of these files will only take effect for the current shell once you `source` the file.

Environment Variables

- *Environment variables* are used to pass information to other utility programs; they do get passed on to new processes.
- E.g., the environment variable `MANPATH` tells the `man` command where it should look for manual pages.
- In a `bash` environment, the `export` command is used to set an environment variable:
  `export TGIFPATH=/usr/local/openwin/lib/tgif`.
  Note that there should not be any space between the variable and the equal sign and the value.
- In a `tcsh` environment, environment variables are set by the `setenv` command and unset by the `unsetenv` command. E.g.,
  `setenv TGIFPATH /usr/local/openwin/lib/tgif`.
- The value of an environment variable is obtained by prefixing it with “$”.
- You can output the value of an environment variable by using the `echo` command: `echo $TGIFPATH`.

Environment Variables

- The `env` command lists all environment variables currently set.
- Some of the most important environment variables include:

HOME	your home directory; e.g., `/home/cowi/held`.
TERM	your terminal type; e.g., `xterm` or `vt100`.
SHELL	the path to your shell; e.g., `/bin/tcsh` or `/bin/bash`.
USER	your login name; e.g., `held`.
PATH	a list of directories to search for commands.

- The `TERM` variable describes the characteristics and basic functionality of a terminal, such as how to write characters to the screen, move the cursor, etc.

Setting the `PATH`

- The `PATH` environment variable contains a list of directories that your shell will look in for commands.
- Commands on a Unix system are simply executable files, usually compiled from C source code, or (shell) scripts. (To be precise, every shell also sports built-ins.)
- The shell looks through all directories in `PATH` when it first starts up and builds an internal list of executable files available in the directories.
- For instance, the `rm` command typically resides in `/usr/bin` or `/bin`, but if your `PATH` variable does not contain `/usr/bin` or `/bin` then you will not be able to execute `rm`. (Unless you'd specify the full path, that is `/usr/bin/rm` or `/bin/rm`.)
- If an executable gets added to one of those directories after you started a `tcsh` shell then you need to use the `rehash` command to have the shell rebuild its internal hash table.
- In a `tcsh` environment, you can discard this list by using the `unhash` command.
- In a `bash` environment no full hash table is built and, thus, the `rehash` command is not needed and not available. Use `hash -r` if you used a program in the current shell and later moved it elsewhere.

Setting the `PATH`

- Normally, sysadmins will put a reasonable setting for `PATH` in a system-wide startup file.
- The following entry in a `bash` environment is typical for a basic user-defined set-up:

  ```
  export PATH=${HOME}/bin:/usr/bin:/usr/local/bin:${PATH}:.
  ```

  In a `.cshrc` file, for a `tcsh` environment:

  ```
  setenv PATH ${HOME}/bin:/usr/bin:/usr/local/bin:${PATH}:.
  ```

  It instructs the shell to search the subdirectory `bin` of your home as the first directory, followed by `/usr/bin` and `/usr/local/bin`. In addition, any directories specified by a pre-defined (system-wide) `PATH` variable are searched. Finally, your current working directory is searched.
- Note that there is some controversy about whether or not to include your current working directory in the search path!
- Allowing the implicit use and execution of files within the current working directory makes you vulnerable to “Trojan-Horse” attacks in hostile environments.
- And, after all, you can always execute a binary `foo` in your current directory by typing `./foo`!

Shell Variables

- *Shell variables* are strictly internal to the running shell. That is, shell variables are not passed on to another shell or to an application program.
- Shell variables can be regarded like variables in programming languages.
- A typical shell variable is `PWD`, which contains the (absolute) path of the current working directory. (The command `echo $PWD` will display its current value.)
- Some shell variables automatically set the matching environment variable for you. E.g., setting `term` causes the environment variable `TERM` to be set.
- As a general guideline, it is advisable to use environment variables rather than shell variables (if and when environment variables are available) — unless use inside a shell script is intended.

Setting the Prompt

- You can set the `tcsh` shell variable `prompt` to be what you want the prompt to be.
- For `bash`, the respective variable is called `PS1` (shorthand for “prompt statement”).
- The shell will recognize formatting sequences inside the prompt string.
- Some of the more useful formatting sequences available are:

tcsh	bash	
`%m`	`\h`	. . . computer’s name (with full domain specification omitted).
`%n`	`\u`	. . . user’s name.
`%t`	`\@`	. . . time.
`%/`	`\w`	. . . the current working directory.
`\!`	`\!`	. . . the history number of this command.

- For instance, the command `set prompt="\!  %n@%m-> "` defines my personal prompt for `tcsh`: “`79 held@ursus->`”.
- In a `bash` shell, the equivalent command would be `PS1="\!  \u@\h-> "`.

Aliases

- The basic form of an alias command is

  ```
  alias <command> <string>          for (t)csh,
  ```

  and

  ```
  alias <command>=<string>          for bash.
  ```

- E.g., `alias ll 'ls -l'` or `alias ll='ls -l'`.
- When typing “`ll`” followed by pressing “return”, the shell intercepts it – since it is watching for aliases – and *expands* it to “`ls -l`”.
- More generally, the first word of a simple command, if unquoted, is checked to see if it has an alias. If so, that word is replaced by the text of the alias.
- Aliases allow you to replace frequently used long commands by shorter equivalents, to change commands such that they get called with certain flags by default, or to mimic commands that work on other systems.
- When used without options, the `alias` command lists all currently defined aliases.
- A previously defined alias can be removed by means of the `unalias` command.

Aliases

- Some sample aliases (for `tcsh`):

```
alias  del ’rm -i’
alias  dir ’pwd; ls -xF’
alias  lit ’cd \$HOME/papers/biblio/martin; dir’
alias gzip ’gzip -9’
alias ftps ’ftp ursus.cosy.sbg.ac.at’
```

- Some sample aliases (for `bash`):

```
alias  del=’rm -i’
alias  dir=’pwd; ls -xF’
alias  lit=’cd \$HOME/papers/biblio/martin; dir’
alias gzip=’gzip -9’
alias ftps=’ftp ursus.cosy.sbg.ac.at’
```

Aliases That Take Arguments

- For `tcsh`:
  - Use `\!\$` to indicate that the last command line argument should be appended to alias, and `\!:`*n* to specify the *n*-th argument.
  - Use `\!*` to indicate that all command line arguments should be appended to alias instead of just last one.
  - Some sample aliases (for `tcsh`):

```
alias sd         'cd \!*; dir'
alias addheader 'cat ~/code/my_header.txt \!:1 > \!:2'
alias search     'find . -name "\!*" -print'
alias trash      'mv \!* ~/Trash'
```

- For `bash`:
  - A `bash` alias does not accept parameters directly; you will have to create a function.
  - The arguments are specified in the same way as for `sh` shell scripts.
  - E.g., for `bash`:

```
sd()          { cd $1; dir; }
addheader() { cat ~/code/my_header.txt $1 > $2; }
search()      { find . -name "$1" -print; }
trash()       { mv $@ ~/Trash; }
```

  - Observe all spaces and don't forget the semi-colon before the closing right curly brace!

Sample .cshrc File

```
# Set file permissions.
#
umask 077

# Use $prompt for key to determine whether we should read these
# aliases. If running a script, $prompt # will be unset.
#
if ( $?prompt ) then
    # Start aliases for interactive work
    #
    alias del     "rm -i"
endif

# Grab any local .cshrc file
#
if ( -r ~/.cshrc.loc ) then
    source ~/.cshrc.loc
endif
```

Sample `.profile` File

```
# Uncomment for debugging purposes:
# echo "HOME/.profile"

# Get aliases and functions
if [ -f ~/.bashrc ]; then
  . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:/usr/local/sw/jdk/bin:$HOME/bin

export PATH
```

Sample `.bashrc` File

```
# Uncomment for debugging purposes:
# echo "HOME/.bashrc"

# Get system-wide aliases first
if [ -f /etc/bashrc ]; then
  source /etc/bashrc
fi

alias del="rm -i"
```

Other Configuration Files

- Most utilities can be configured and customized to one's liking by means of utility-specific configuration files. E.g.,

`.acrorc`	configuration file for `acroread`;
`.emacs`	configuration file for `emacs`;
`.mozilla`	configuration directory for `mozilla`;
`.thunderbird`	configuration directory for `mozilla-thunderbird`.

- Unless the default name and location is used, an appropriate environment variable has to be set in order to tell the utility where to look for its configuration file.
- It is one of the harsh facts about customization that pretty much every utility has its own idiosyncratic customization syntax.

Advanced Unix and Shell Programming

- Unix puts powerful tools at your finger-tips, some of which we will explore on the subsequent slides.
- As well as using the shell to run commands you can use its built-in programming language to write your own commands or programs.
- You can put commands into a file – known as a *shell script* – and then execute that file as you would execute a program.
- It is widely recommended to use the Bourne shell (`sh`) for shell scripts.
- Note that `sh` shell scripts will also work in a `bash` environment.
- The syntax of `csh` and `tcsh` scripts is different, though!

Simple Shell Script

```
#!/bin/sh
#
# This script displays the date, time, username and current
# directory.
#
echo "Date and time is:"
date
echo
echo "Your username is: `whoami`"
echo "Your current directory is: `pwd`"
```

- Lines 2–5, which begin with a hash “#”, are comments and are not interpreted by the shell.
- Use comments to document your shell script; you will be surprised how easy it is to forget what your own programs do!

Simple Shell Script Dissected

- To make sure that your shell script is run from a standard Bourne shell, always have the line
  `#!/bin/sh`
  as the first line of the script. (A line that starts with `#!` is called *shebang*.)
- The back-quotes around the command `who am i` illustrate the use of command substitution.
- The argument to the `echo` command is quoted to prevent the shell from interpreting it.

Passing Arguments to a Shell Script

- Shell scripts can act like standard Unix commands and take arguments from the command line.
- Arguments are passed from the command line into a shell program using the *positional parameters* `$1` through to `$9`.
- Each parameter corresponds to the position of the argument on the command line.
- The positional parameter `$0` refers to the command name or name of the executable file containing the shell script.
- Only nine command line arguments can be accessed in this straightforward way; additional arguments can be accessed by using the `shift` command.
- All the positional parameters can be referred to using the special parameter `$*`. (This is particularly useful when passing filenames as arguments.)

Passing Arguments to a Shell Script

```
#!/bin/sh
# This script converts ASCII files to PostScript
# It uses a local utility "a2ps"
#
echo "usage: ascii2ps <ascii-file> <postscript-file>"
a2ps --columns=1 --portrait $1 > $2
```

- The `shift` command gives access to command line arguments greater than nine by shifting each of the arguments.
- The second argument (`$2`) becomes the first (`$1`), the third (`$3`) becomes the second (`$2`), and so on.
- Successive `shift` commands make additional arguments available.
- Note that there is no "`unshift`" command!

Handling Shell Variables

- The shell has several variables which are automatically set whenever you login.
- Any name defined in your environment can be accessed from within a shell script.
- Special shell variables (command-line parameters):

`$1` – `$9`	these variables are the positional parameters.
`$0`	the name of the command currently being executed.
`$#`	the number of positional arguments given.
`$?`	the exit status of the last command executed. 0 (zero) if successfully completed, non-zero otherwise.
`$$`	the PID of this shell.
`$!`	the PID of the last command run in the background.
`$-`	the current options supplied to this invocation of the shell.
`$*`	a string containing all the arguments to the shell, starting at $1.

Command-Line Parameters

```
#!/bin/sh
#
# command-line parameters
#
echo "Number of command-line parameters $#"
echo "List of command-line parameters: $*"
echo "Parameter #1: $1"
echo "User: $USER"
echo "Shell: $SHELL"
echo "File name of shell script: $0"
echo "PID: $$"
a=17.89        # without a blank around "="!
echo "A set to $a"
```

Evaluating Shell Variables

- The following set of rules governs the evaluation of all shell variables.

`$var`	signifies the value of `var` or nothing, if var is undefined.
`${var}`	same as above except the braces enclose the name of the variable to be substituted.
`${var-thing}`	if defined, $`var`; otherwise `thing`.
`${var=thing}`	if defined, $`var`; otherwise `thing`. if undefined, $`var` is set to `thing`.
`${var?message}`	if defined, $`var`; otherwise print `message` and exit the shell.
`${var+thing}`	`thing` if $`var` is defined; otherwise nothing.

Variable Referencing

```
#!/bin/sh

a=37.5
hello=$a
# No space permitted on either side of = sign.

echo "hello"

echo $hello
echo ${hello}     #Identical as above.

echo "$hello"
echo "${hello}"

echo '$hello'
# Variable referencing disabled by single quotes.
# Notice the effect of different types of quoting!
```

Reading User Input

- To read standard input into a shell script use the `read` command. E.g.,

```
echo "Please enter your name:"
read name
echo "Welcome back, $name!"
```

- If there is more than one word in the input, each word can be assigned to a different variable. Any words left over are assigned to the last named variable. E.g.,

```
echo "Please enter your surname\n"
echo "followed by your first name:  \c"
read name1 name2
echo "Welcome back, $name2 $name1"
```

Conditional Statements

- Every Unix command returns a value on exit which the shell can interrogate. This value is held in the read-only shell variable `$?`.
- The `if` statement uses the exit status of the given command and conditionally executes the statements following.
- The words `then`, `else` and `fi` are shell-reserved words and as such are only recognized after a newline or semicolon ";".
- Make sure to end every `if`-construct with a `fi` statement.
- You can use the && operator to execute a command and, if it is successful, execute the next command in the list.
- You can use the || operator to execute a command and, if it fails, execute the next command in the command list.

Conditional Statements

```
#!/bin/sh
# usage: available.sh username
if who | grep $1 &> /dev/null
then
echo "$1 is logged on to this machine"
else
echo "$1 is not available"
fi
```

- This lists `who` is currently logged on to the sytem and pipes the output through `grep` to search for the username.
- The `-s` option causes `grep` to work silently; any error messages are directed to the file `/dev/null` instead of the standard output.

Conditional Expressions

"`s1`" = "`s2`"	true if the strings `s1`,`s2` are identical
"`s1`" != "`s2`"	true if the strings `s1`,`s2` are not identical
-z "`s1`"	true if string `s1` is empty (i.e., has length zero)
-n "`s1`"	true if string `s1` is not empty
`n1` -eq `n2`	true if the numbers `n1`,`n2` are identical
`n1` -ne `n2`	true if the numbers `n1`,`n2` are not identical
`n1` -gt `n2`	true if `n1` is greater than `n2`
!	negation
-a	logical and
-o	logical or
-e `<file>`	true if `<file>` exists
-f `<file>`	true if `<file>` is a regular file (no subdirectory or symbolic link)
-r `<file>`	true if `<file>` is readable

- The shell provides the `test` command to evaluate conditional expressions. See its manual page for details.
- The command `[ <cond> ]` may be used as an alias for `test <cond>`.

Conditional Expressions

```
#!/bin/sh
#
echo "hello. answer Y to proceed:"
read answer
echo "your answer was" $answer
if test "$answer" != "n"
  then echo "yes"
  else echo "no"
fi
```

More Control Statements: Case

```
case word in
   pattern1) command(s)
   ;;
   pattern2) command(s)
   ;;
   patternN) command(s)
   ;;
esac
```

- The `case` statement provides for multi-way branching based on patterns.
- When all the commands are executed control is passed to the first statement after the `esac` statement. Every list of commands must end with a double semicolon “;;”.
- Patterns are checked for a match in the order in which they appear.
- The asterisk “$\star$” can be used to specify a default pattern.

More Control Statements: For

```
for var in list-of-words
do
   commands
done
```

- Here, `commands` is a sequence of one or more commands separated by a newline or a semicolon.
- The reserved words `do` and `done` must be preceded by a newline or a semicolon.

Sample For-Loop

```
#!/bin/sh
# see if a number of people are logged in
for i in $*
do
    if who | grep -s $i > /dev/null
    then
    echo "$i is logged on to this machine"
    else
    echo "$i is not available"
    fi
done
```

More Control Statements: While

```
while command-list1
do
    command-list2
done
```

- The commands in `command-list1` are executed; and if the exit status of the last command in that list is 0 (zero), then the commands in `command-list2` are executed.
- The loop is executed as long as the exit status of `command-list1` is 0 (zero).

More Control Statements: Until

```
until command-list1
do
    command-list2
done
```

- Identical in function to the `while` command except that the loop is executed as long as the exit status of `command-list1` is non-zero.

Another Sample Script

```
#!/bin/sh
case $# in
   1) ;;
   *) echo "usage: watch.sh username" ; exit 1
esac
until who | grep -s "$1" >/dev/null
do
   sleep 60
done
echo "$1 has logged in"
```

Break and Continue Statements

- The `break` command terminates the execution of the innermost enclosing loop, causing execution to resume after the nearest done statement.
- To exit from *n* levels, use the command `break n`.
- The `continue` command causes execution to resume at the `while`, `until` or `for` statement which begins the loop containing the `continue` command.

More Shell Functions

- The shell does not have any arithmetic features built in and so you have to use the `expr` command. See the manual pages for details.
- Another built-in function is `eval` which takes the arguments on the command line and executes them as a command.
- The `exec` statement causes the command specified as its argument to be executed in place of the current shell without creating a new process.
- The `exit` statement will exit the current shell script. It can be given a numeric argument which is the script's exit status. If omitted the exit status of the last run command is used.
- Shell scripts may use the `trap` command to catch (or ignore) OS signals. Again, see the manual pages for details.

Debugging Shell Scripts

- To see where a script produces an error use the command

  `sh -x <script_name> <arguments>`.
- The `-x` option to the `sh` command tells it to print commands and their arguments as they are executed.
- Other debugging options include

`-e`	in non-interactive mode, exit immediately if a command fails.
`-v`	print shell input lines as they are read.
`-n`	read commands but do not execute them.

Regular Expressions and Stream Editor

- A regular expression is a sequence of characters that forms a template used to search for strings within text.
- E.g., `grep -w 't[a-i]e'` matches the words tee, the or tie. The brackets have a special significance. They mean to match one character that can be anything from “a” to “i”.
- Similar, `grep -w 'cr[a-m]*t'` matches the words credit, craft or cricket. The “*” means to match any number of the previous characters, which in this case is any character from “a” through “m”.
- The `grep` command works nicely in conjunction with the *stream editor*, `sed`. In the way that `grep` can search for words and filter lines of text; `sed` can do search-replace operations and insert and delete lines into text.
- See the online documentation for details.

Communication and Data Transfer

- Internet
- Remote Access, ftp, E-Mail
- WWW

What is the Internet?

- The Internet is the largest computer network in the world.
- The Internet is the back-bone connection of local, national and international networks which are locally administrated (domains).
- It connects more than 10^9 (computing) devices.
- Communication within the Internet is based on a standardized protocol (TCP/IP).
- The Internet supports a variety of services, e.g., telnet, ftp, e-mail, Usenet news, IRC, and WWW.
- The Internet is not to be confused with its supported service WWW.

History of the Internet (–1979)

- 1957: USSR launches Sputnik, the first artificial earth satellite.
- 1958: The Sputnik Shock motivates the US Department of Defense to found ARPA (Advanced Research Project Agency). Goal: world-wide military leadership for the USA.
- 1962–1968: Concepts of ‘packet-switching-networks’ are developed.
- 1969: The first ’Interface Message Processor’ is installed at UCLA. Four sites (UCLA, UCSB, SRI, U. of Utah) form the first nodes of the ARPANET.
- 1971: Electronic mail (email) invented.
- 1972: First public demonstration of ARPANET among 40 machines.
- 1973: Metcalfe’s Harvard PhD Thesis outlines idea for Ethernet.

History of the Internet (1980–1989)

- 80's: Several other networks spawn off ARPANET: e.g., BITNET (Because It's Time NETwork), CSNET (Computer Science NETwork), MILNET (Military Network).
- 1982: TCP/IP established by ARPA as protocol suite for ARPANET.
- 1983: IAB (Internet Activities Board) established; EARN (European Academic and Research Network) established.
- 1984: Number of hosts breaks 1 000; Domain Name System (DNS) introduced.
- 1986: First meeting of IETF (Internet Engineering Task Force).
- 1988: Internet worm burrows through the Internet, affecting about 6 000 of the 60 000 hosts on the Internet; CERT (Computer Emergency Response Team) formed by DARPA in response to the worm.
- 1989: Number of hosts breaks 100 000.

History of the Internet (1990–1999)

- 1990: 3 000 networks; number of hosts breaks 300 000; ARPANET ceases to exist.
- 1991: World-Wide Web (WWW, developed by Berners-Lee) released by CERN; PGP (Pretty Good Privacy) released by Zimmerman; Internet Society founded.
- 1992: Number of hosts breaks 1 000 000; Internet Activities re-organized and re-named the Internet Architecture Board.
- 1993: Al Gore's initiative (in the early 90's) for an "Information Super Highway" leads to Internet boom; White House (`.gov`) and and the United Nations (`.org`) go online; Mosaic becomes first graphical web browser; Internet Engineering Task Force (ITEF) turns international under auspices of Internet Society.
- 1994: Netscape Navigator.
- 1995: New WWW technologies emerge: mobile code (Java, JavaScript), virtual environments (VRML); Traditional online dial-up systems (Compuserve, AOL) begin to provide Internet access; Amazon online.
- 1996: HoTMaiL provides the first webmail service.
- 1998: IPv6 protocol defined to replace IPv4; Google search engine; Internet Corporation for Assigned Names and Numbers (ICANN) incorporated.
- 1999: IEEE 802.11b wireless networking.

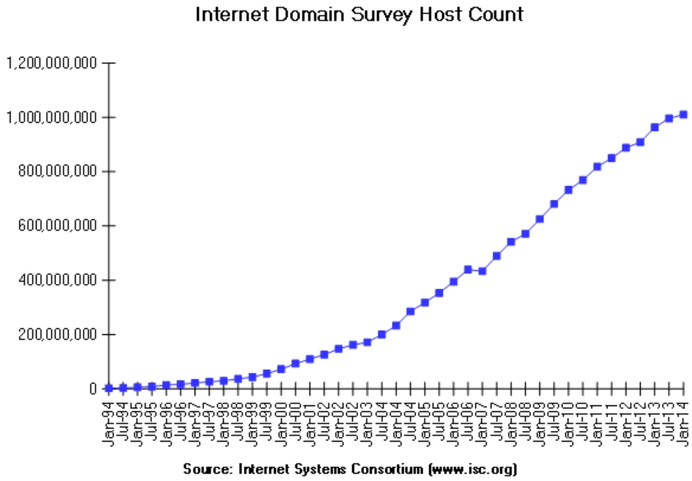
History of the Internet (2000–Present)

- 2000: Dot-com bubble bursts.
- 2001: New top-level domains active (aero, biz, coop, info, name, museum, pro); Wikipedia online.
- 2003: Skype makes VoIP widely available.
- 2004: Facebook, Flickr, Podcast; Web 2.0.
- 2005: YouTube.
- 2006: Twitter.
- 2007: WikiLeaks; iPhone makes its debut.
- 2008: Dropbox.
- 2010: First unassisted live Internet link into low earth orbit.
- 2012: ICANN starts accepting proposals for new generic top-level domain names (gTLD).
- 2013–2015: Many new gTLDs launched, for a total of more than 220 gTLDS.

- Widespread use and growth: Blogging, gaming, video streaming, social networking, cloud computing/storage, VoIP (Voice over Internet Protocol).

Growth of the Internet

- Aug 1981: 213 hosts; Oct 1984: 1024 hosts; Jan 2000: 72 398 092 hosts; July 2015: 1 033 836 245 hosts. (Credit: Internet Systems Consortium.)



TCP/IP

- TCP/IP (Transmission Control Protocol / Internet Protocol) is the suite of communication protocols used to connect hosts on the Internet.
- TCP/IP uses several protocols, the two main ones being TCP and IP.
- TCP/IP is built into the Unix OS, making it the de-facto standard for transmitting data over networks.
- Basically, IP handles the flow of data between host computers within the Internet.
- TCP manages the flow of data between applications.
- Information is transmitted using packets. Data is split up by the sender and combined at the destination.

IP Addresses

- *IPv4 addresses* in the *dotted-decimal* format are unique sets of four period-delimited octets that represent individual hosts on specific networks.
- An IP octet is a number between 0 and 255. Thus, an IP address might look something like 141.201.2.2.
- The range 0 to 255 per octet implies that 4.3 billion ($2^{32} - 1$) IP addresses could exist.
- For technical and administrative reasons, the actual IP address space contains less than 4.3 billion addresses.
- Shortage of IPv4 addresses: The Asian regional Internet registry, APNIC, imposed restrictions on the allocation of new addresses in April 2011. On 14-Sept-2012 RIPE NCC, which serves Europe, started to distribute IPv4 addresses from its last block of addresses.
- Since the current address space will not be able to sustain a continued rapid growth of the Internet, IPv6 (which is short for “Internet Protocol Version 6”) has been designed by the IETF as the “next generation” protocol to replace the current IPv4.

IPv6

- *IPv6* uses 128 address bits and thus allows 2^{128} IP addresses. For human readability the numbers are split in eight blocks of four hexadecimal digits each, separated by a colon, like: 2001:0db8:85a3:08d3:1319:8a2e:0370:7344.
- In addition, the IPv6 protocol supports multicast (sending data to different hosts simultaneously), automatic network configuration of hosts, mobility of hosts, multi-homing (a host has a number of different addresses at the same time).
- IPv6 is supported by almost all modern operating systems but service providers still are slow in migrating to IPv6.
- Hosts can utilize IPv4 and IPv6 at the same time ("dual-stack operation") and the operation is transparent to the end user.

Alternate Formats for IP Addresses

- Alternate “dotless” formats for IP addresses exist, such as *dword* (essentially two 16-bit binary words, base-10), *octal* (base 8), and *hexademical* (base 16).
- E.g., the dotted-decimal IP address 141.201.2.2 translates to 2378760706 in dword-format:
  $2378760706 = 141 \cdot 16777216 + 201 \cdot 65536 + 2 \cdot 256 + 2 \cdot 1.$
- Dotless IP calculators can be found in the WWW. See, for instance, `http://www.rootsolutions.com/tools/dotless/`.
- Note that some browsers, firewalls, and proxy servers might not be able handle the dotless IP addresses.
- Also, note that con artists have recently discovered dotless IP addresses as a new way to obscure and disguise their identity. Stay alert!

IP Data Transmission

- Besides defining the address scheme, IP also handles the transmission of data from the sending computer to the computer specified by the IP address.
- Large, unwieldy chunks of data are broken into easily manageable IP packets that can be delivered across the network.
- Every packet gets its own destination address, check sum, and other information.
- Packets are routed individually across the network and may reach their destination on different routes. The sender does not need to specify (or even know) a route.
- Every time a packet arrives at an IP router, the router decides where to send it next.
- Routers can send data along the path of least resistance, taking into consideration local network traffic congestion.

IP Data Transmission

- If a connection line on the network breaks down, traffic can still reach its destination along an alternative path.
- Recent studies have sparkled a debate on how vulnerable the Internet is to a concerted attack on some of its “main” nodes and links.
- Protocols like IP are called “connection-less” protocols.
- Contrary to a “connection-oriented” protocol, there is no concept of a session with a pre-selected path for all traffic.
- Routing data in a dynamically changing environment is a challenging scientific problem.

Domain Name System

- A *domain* is a group of computers and devices on a network that are administered as a unit with common rules and procedures.
- Domains are defined by their (range of) IP addresses. All devices sharing a common part of the IP address are said to be in the same domain. E.g., the computers within our network have IP addresses of the form 141.201.*xxx*.*yyy*.
- IP addresses are cumbersome for humans to handle. Thus, symbolic names for computers ("host names") were introduced and the *Domain Name System* (DNS) created.
- Every host name must be unique within a domain.
- The typical structure of an academic host name is mycomputer.mydept.myuniversity.mynetwork.mycountry.
- For instance, the host name `ursus.cosy.sbg.ac.at` translates to the IP address 141.201.12.95.

Domain Name

- Domain name: Every domain name has a suffix that indicates which top-level domain (TLD) or country-code top level domain (ccTLD) it belongs to. Up to very recently, there have been seven traditional “generic” TLDs,
  - com – commercial business,
  - edu – US and Canadian educational institutions,
  - gov – US government agencies,
  - int – International,
  - mil – US Military,
  - net – Network organizations,
  - org – Organizations (non-profit),

  one infrastructure TLD (arpa), and about 250 ccTLDs, such as
  - at - Austria,
  - ca - Canada,
  - de - Germany.
- Due to a shortage of domain names at the top level, seven new top-level domains were introduced in 2001: aero, biz, coop, info, name, museum, pro.
- More new top-level domains introduced since 2013.

Domain Name Resolution

- IP addresses are cumbersome for humans to handle. Thus, symbolic names for computers ("host names") were introduced and the *Domain Name System* (DNS) created.
- The DNS is an Internet service that translates host names into IP addresses.
- The DNS is based on a hierarchical tree structure. Below the root are the top-level domains, the second-level domains and eventually further sub-level domains, and finally the host name.
- If a DNS server does not know how to translate a particular host name, it asks another DNS server, and so on, until the correct IP address is returned.
- The command `host` can be used to obtain the IP address of a device specified by its host name.
- E.g., `host www.cosy.sbg.ac.at` returns the IP address `141.201.2.14`.
- Reverse lookups: A DNS-Server gets an IP address and returns the name. E.g., `host 141.201.2.14` returns the name `bulldogge.cosy.sbg.ac.at`, which is an alias for `www.cosy.sbg.ac.at`.
- In order to cut down on spam emails some mailers employ a reverse lookup. (But a reverse lookup should never be used for authentication purposes in a security-relevant application!)

Remote Access

- In networks, *remote* refers to files, devices, and other resources that are not connected directly to your workstation. Resources at your workstation are considered *local*.
- There is no principal difference between remote access to a machine within one's own local-area network (LAN)and remote access to a machine in the Internet (outside of one's LAN).
- *Remote control* refers to taking control of another computer remotely, i.e., via the network.
- *Remote access*, also called *remote login*, means that you are logged into a remote computer that allows you to use it as you would use your own workstation — up to the limits imposed by the utility that performs the remote access.
- Remote-access software is required in order to connect to a remote machine in a network. (And, or course, one needs to have the right to access the remote machine.)
- In a normal setting, the only difference between a remote host and a workstation in front of you is a (possibly) lower rate of data transfer.

Remote Access via Telnet

- *Telnet* is a terminal emulation program for TCP/IP networks.
- The `telnet` program runs on your computer and connects your computer to a host computer.
- To start a telnet session, you must login to the host computer by entering a valid username and password.
- Once a connection has been established, you can then enter commands through the `telnet` program and they will be executed as if you were entering them directly on the server console.
- Telnet is based on a low-level protocol and does not support the exchange of graphics or similar complex data.

Security Hazard

The entire data transfer (including the user authentification) is carried out without encryption!

- Hence, for security reasons, the `telnet` command often is disabled.

Remote Access via Rlogin

- The `rlogin` utility establishes a remote login session from your workstation to a remote machine.
- It offers a slightly higher level of functionality than `telnet` but it is not considered to be safer.

Security Hazard

Similar to `telnet`, `rlogin` exchanges plain-text passwords (without using any encryption).

- Thus, for security reasons, the `rlogin` command often is disabled. If enabled, its use is discouraged; you should use `ssh` instead!

Remote Access via Secure Shell

- The *secure shell*, `ssh`, was developed by SSH Communications Security Ltd.
- It is a program to log into a remote host over a network, to execute commands on the remote host, and to move files from one machine to another. It provides strong authentication and secure communication over insecure channels.
- `Ssh` is a replacement for `rlogin`. When using ssh's login client `slogin`, instead of `rlogin`, the entire login session, including the transmission of the password, is encrypted.
- Thus, when using `ssh` it is almost impossible — or, at least, considerably more difficult — for an outsider to collect passwords.
- Both host-based and certificate-based authentication is possible.
- Sample use:
  `ssh held@sshstud.cosy.sbg.ac.at`.
- Of course, the IP address or full Internet name of the remote machine has to be known.
- The `ssh` utility is widely regarded as the only safe way to access a remote host.
- X11 forwarding is enabled by `ssh -X <hostname>`. However, use this feature with caution, and do not use it if the remote host is not entirely trustworthy!

Data Transfer via Secure Copy

- Copying files between machines in a network can be achieved by using *secure copy*, `scp`.
- `Scp` uses `ssh` for data transfer, and uses the same authentication and provides the same level of security as `ssh`.
- It replaces the command `rcp`, which is based on `rlogin`.
- Sample use:
  `scp foo.txt held@krontaube.cosy.sbg.ac.at:/home/cowi1/held`.
- Note that `scp` requires one to have an active user account on the remote machine.

Data Transfer via FTP

- Another protocol used for exchanging data within the Internet is called ftp (File Transfer Protocol).
- In order to be able to transfer data via `ftp` between a local host and a remote computer,
  - both the local host and remote host need to support `ftp`,
  - the IP address or full Internet name of the remote host has to be known,
  - an active account on both machines is needed,
    or one is restricted to “anonymous ftp”.
- To initiate an ftp data transfer, use the command `ftp <host_name>` or, if the IP address is known, `ftp <host_IP_address>`.
- FTP does not encrypt its traffic and is prone to sniffing.
- If supported by both machines, you may want to resort to `sftp` in order to rely on a secure `ssh` transport of your ftp session.

Data Transfer via FTP

- Once the ftp session has been established, use

  ```
  put <local_file> <remote_file>
  ```

  and

  ```
  get <remote_file> <local_file>
  ```

  to send or receive a file.
- Sample ftp session:

  ```
  ftp > cd rfc
  250 CWD command successful.
  ftp > bin
  200 Type set to I.
  ftp > get rfc1048.txt.gz
  200 PORT command successful.
  150 Binary data connection for rfc1048.txt.gz (5141 bytes).
  226 Transfer complete.
  ftp > quit
  221 Goodbye.
  ```

Data Transfer via FTP

- By default, `ftp` assumes the file being transferred is an *ASCII file*.
- You may need to use the command `binary` within the ftp session to instruct `ftp` to handle a *binary* file, such as an executable or a compressed file.
- Newer ftp clients attempt to determine whether the ftp server is of the same system type, and, if so, transfer all files in binary instead of ASCII mode.
- Transferring ASCII files in binary mode can be expected to reduce the transmission time by decreasing the size of the data transferred and by increasing the speed of the data exchange between the sender and the receiver.
- Use *passive mode*, by specifying `passive`, if the ftp server cannot establish a connection to the ftp client. (E.g., if the network of the client is protected by a firewall.)
- The option `-d` can be used for running `ftp` in debug mode, which causes it to see the client commands in addition to the server's reponses.

Data Transfer via FTP

Possible Error Messages:

- Unknown host: The hostname you specified was not found, or does not exist. Check the spelling of the hostname and try again.
- Foreign host did not respond within OPEN timeout: Either the specified host does not support ftp, or the host is busy and could not open an ftp connection. Try again later.

Keep in mind:

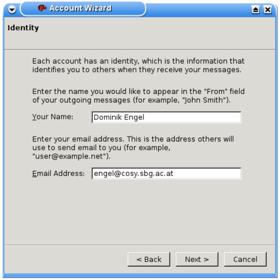
- Enter all ftp commands in lower case. (Unix machines are case-sensitive!)
- Type file names exactly as they are listed.
- Binary versus ASCII files – in case of doubt, use binary mode!
- For security reasons, keep all ftp sessions as brief as possible.

Anonymous FTP

- *Anonymous ftp* is the mode of operation used for accessing public files on remote machines.
- To use anonymous ftp, start a conventional ftp session, and then enter the word “anonymous” or “ftp” when the host prompts you for a username; you can enter anything for the password.
- Basically, this means that you do not need to identify yourself before accessing files.
- However, please note that most ftp servers keep log files of all ftp transactions!
- Needless to say, accessing remote files via anonymous ftp is only possible if the remote machine allows such an access.
- Please make sure to select an ftp server that is geographically close to your site when downloading software.
- When downloading software via anonymous ftp you should always check whether mirror sites exist that are closer to you than the original ftp server!

Electronic Mail: Thunderbird

- Mozilla Thunderbird allows you to read e-mail and news.
- It supports POP (Post Office Protocol) and IMAP (Internet Message Access Protocol).
- It has an address book to manage contacts.
- Thunderbird offers decent spam control. (Spam is unsolicited bulk commercial email.)
- The initial configuration has been carried out during the lab sessions.



Identity

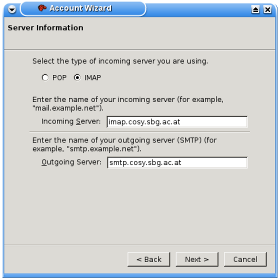
Each account has an identity, which is the information that identifies you to others when they receive your messages.

Enter the name you would like to appear in the "From" field of your outgoing messages (for example, "John Smith").

Your Name: Dominik Engel

Enter your email address. This is the address others will use to send email to you (for example, "user@example.net").

Email Address: engel@cosy.sbg.ac.at



Server Information

Select the type of incoming server you are using.

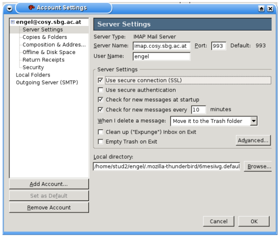
POP [✓ IMAP]

Enter the name of your incoming server (for example, "mail.example.net").

Incoming Server: imap.cosy.sbg.ac.at

Enter the name of your outgoing server (SMTP) (for example, "smtp.example.net").

Outgoing Server: smtp.cosy.sbg.ac.at



Server Settings

Server Type: IMAP Mail Server

Server Name: imap.cosy.sbg.ac.at Port: 993 Default: 993

User Name: engel

Server Settings

- [✓] Use secure connection (SSL)
- [ ] Use secure authentication
- [✓] Check for new messages at startup
- [✓] Check for new messages every 10 minutes

When I delete a message: Move it to the Trash folder

- [ ] Clean up ("Expunge") Inbox on Exit
- [ ] Empty Trash on Exit

Local directory: /home/stud2/engel/.mozilla-thunderbird/6mesiivg.defaul

- Under Debian and Ubuntu Linux, Firefox and Thunderbird are called Iceweasel and Icedove.

Security Issues: Thunderbird

- Its password manager allows to save usernames and passwords for your e-mail accounts.

Security Warning

The way in which Thunderbird stores these passwords is **not** secure! (Just like in the case of the Mozilla/Firefox browser.)

- Thus, you should not use this feature unless you are in a secure environment.
- Alternatively, please use a master password!
- You can delete saved passwords in Firefox and Thunderbird as follows:
  `Edit` → `Preferences` → `Privacy` → `Passwords` → `View Saved Passwords`
  → `Remove All`
- See also: https://rtfm.cosy.sbg.ac.at/doku.php

Security Issues

- A *firewall* filters (incoming) packets: Only packets matching specific rules are allowed to pass on to the network. Rules might be: E-mail allowed, telnet only to specific computers, exclude certain web servers.
- PGP (Pretty Good Privacy) uses a two-key *encryption* method: one key is private, the other key public. Everyone may use the public key to encode a message; the private key is used to decode the message.
- PGP may be used for *authenticity checks*: Using the private key to encrypt the signature, the signature may be decoded with the public key. If the info in the signature is the same, an unique identification of the sender is possible.
- GPG (GnuPG) is the GNU-Version of PGP. `gpg` is a commandline tool for key-management, encryption, decryption, signing and verifying. There also exist graphical front-ends for GPG.
- Enigmail is a plugin for Thunderbird that allows you to use GPG integrated into Thunderbird.

Basics of the WWW

- The World-Wide Web (WWW) is a service of the Internet. It is used to present information in a hardware-independent way.
- Information is coded in a hypertext structure.
- Software used to “display” the information is called a browser.
- Information access is realized using the client/server principle (browser/web server).
- Most web browsers also support ftp and email.

Basics of the WWW

- A *Web page* is a document that uses the HTML-format (Hyper Text Markup Language).
- A *home page* provides the start page of your web browser when it is launched, e.g., your personal page.
- A *Web server* is a computer that uses specific protocols to support the transport of data related to Web pages within the Internet.
- *HTTP* (Hyper Text Transport Protocol) is the protocol for the transport of hyper text pages (such as HTML-pages).
- The client program (web browser) needs to know the *URL* (Uniform Resource Locator) of a content (e.g., webpage) in order to obtain it from the server. The URL may include information on the protocol, the address of the server (host name), user name, password, port number, directory and filename of the file.

History of the WWW

- In 1991, Tim Berners-Lee invented the WWW at CERN. Its original goal was to facilitate the publication of internal documents.
- In 1993, the WWW-browser Mosaic by NCSA becomes available.
- In 1996, the first international conference on WWW is held in Geneva.
- In the mid 90's, Hyper-G is introduced by the University of Graz. Hyper-G allows to follow every link in both directions.
- Adobe's PDF facilitates the publication of print-quality documents on the Web.
- Around 1995, the Web supports 3D graphics with the advent of VRML (Virtual Reality Modeling Language).
- In 1997, HTML 4 was standardized.
- Around the year 2000, trends like CSS (Cascading Style Sheets) and animated web pages (e.g., Flash, Shockwave) became popular on the client-side web development. On the server side, techniques and languages like PHP and Java servlets made web pages dynamic.
- Strong interaction of users with webpages like youtube, facebook, studivz, delicous and the like, and technologies like wikis and application-like behaviour of webpages led to the vague term of "Web 2.0" in recent years.
- In October 2014, the World Wide Web Consortium (W3C) provided the full specification of HTML5.

Web Communication: Viewing a Webpage

- From the user's point of view:
  - The user starts the browser.
  - The user types in the URL (Uniform Resource Locator).
  - The Web page is displayed.
- From the computer's point of view:
  - The client process (i.e., the web browser) makes a request.
  - The web server responds.
  - The transaction is done.

Web Communication in Detail

- When the client sends a request, the first data item it specifies is an HTTP command that tells the server the type of action it wants to perform.
- This first line of a request also specifies the address of a document (URL) and the version of the HTTP protocol it is using. E.g., `GET /intro.html HTTP/1.0`.
- The server processes the request and sends a response. The first line is the status line. E.g., `HTTP/1.0 200 OK`, which means that the request was successful; 404 means page not found, 403 means access permission denied.
- When a client connects to a server and makes an HTTP request, the request can be of several different types, called methods.
- The most frequently used methods are `GET` and `POST`.
- Basically, the `GET` method is designed for getting information (e.g., a document, a chart, or the result from a database query).
- The `POST` method is designed for posting information (e.g., a credit card number, some new chart data, or information that is to be stored in a database).

HTTP Status Information and Error Codes

100:	continue
101:	switching protocols
200:	OK
201:	created
202:	accepted
203:	non-authoritative information
204:	no content
205:	reset content
206:	partial content
300:	multiple choices
301:	moved permanently
302:	moved temporarily
303:	see other
304:	not modified
305:	use proxy

HTTP Status Information and Error Codes

400:	bad request
401:	unauthorized
402:	payment required
403:	forbidden
404:	not found
405:	method not allowed
406:	not acceptable
407:	proxy authentication required
408:	request time-out
409:	conflict
410:	gone
411:	length required
412:	preconditions failed
413:	request entity too large
414:	request-URI too large
415:	unsupported media type

HTTP Status Information and Error Codes

500:	server error
501:	not implemented
502:	bad gateway
503:	out of resources
504:	gateway time-out
505:	http version not supported

HTML (Hypertext Markup Language)

- Text processing in an Microsoft environment today (often) is WYSIWYG: “what you see is what you get”.
- However, command-oriented text processing still is wide-spread in Unix environments. (And, typically, it is strongly favored by more experienced Unix users!)
- The roots of modern command-oriented text processing date back to 1970 when Knuth invented TEX. In the early 80’s, Lamport expanded TEX to LATEX.
- Characteristic for both systems is the fact that the logical design (contents) and the physical layout (formating) of a text are defined separately.
- In particular, the content is structured with logical markups like heading, paragraph, table width, etc.
- In the mid 80’s, *SGML* (Standard Generalized Markup Language) and *HTML* (Hypertext Markup Language) were invented. Both languages are command-oriented and use logical markups for structuring text.

Basic HTML Document

```
<HTML>
<HEAD>
    <META CONTENT="text/html">
    <TITLE>window title to be displayed by the browser</TITLE>
</HEAD>
<BODY>
    here comes the actual hypertext...
</BODY>
</HTML>
```

- Note that you can learn programming in HTML by poking around the Web: most browsers allow you to look at the HTML code of a page that interests you!

Common HTML Markup Commands (“Tags”)

- `<H1> Text </H1>`
  creates Level-1 headings;
- `<H2> Text </H2>`
  creates Level-2 (sub)headings;
- `<A HREF="http://www.cosy.sbg.ac.at/"> CoWi </A>`
  creates a link to the URL specified;
- `<CENTER> Text </CENTER>`
  centers the text;
- `<STRONG> Text </STRONG>`
  highlights a text by (typically) setting it in bold-face.
- `<EM> Text </EM>`
  sets the text in emphasized mode (typically in italics).

Recap and Self-Assessment

Recap

- In the following we give some examples of what you should be able to do after this course.
- Make sure that you know how to do each of the listed items after the training sessions.
- While in order to pass the course it is required for you to know all of the following items, the enumeration is not exhaustive (in particular, it focuses on the practical side of your skills, not the theoretical side). I.e., knowing all of these items is necessary to pass, but not sufficient!
- Again: The items on this list are only meant as examples. This is **not** a list of questions for the exam!

File System

- Create and rename directories.
- Use relative and absolute paths to change into a directory.
- List the contents of a directory (in different formats, and understand the output).
- Use wildcards.
- Find files matching a specific pattern.
- Create files.
- Display files on screen.
- Find lines with a specific pattern in a file.
- Move, rename, copy and delete files.

File System

- Create symbolic links to files.
- View/change the permissions of a file/directory.
- View/change user/group ownership of a file/directory.
- Create an archive from a set of files or a directory.
- Compress files.
- Compare two files.
- Sort files.
- Filter files.

Bash

- Get help on any command.
- Show the history of commands you entered.
- Create/view/delete an alias.
- Redirect the output of a command to another command.
- Redirect the output of a command to a file (either overwriting or appending to the file).
- Change your password.
- Using a file as input on standard in for a command.
- Know the difference between environment and local shell variables.
- Know the names and functions of the most important environment variables.
- Know how to change the settings for an environment variable.

Job and Process Control

- Know the difference between a job number and a process-id.
- Display all processes.
- Display all jobs.
- Suspend a running process.
- Put a suspended process to the background.
- Start a process in the background.
- Put a background process to the foreground.
- Kill a process.

Communication and Data Transfer

- Know the client/server concept.
- Know how to resolve a given hostname to an IP address (and vice versa).
- Remotely access a workstation in a secure way.
- Transfer files from and to a remote workstation.
- Know what is meant by anonymous FTP.
- Write an E-Mail.
- Know about security issues concerning mail and web.
- Run an X application remotely.
- Know the most common services available on the Internet.
- Know the basic structure of an HTML-document.

The End!

I hope that you enjoyed this course, and I wish you all the best for your future studies.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab