Unix Fundamentals

Dr. Christoph Bauer

Sep 2018

Preface

- When we talk about "Unix," we usually include "Linux," too.
- The first release of this course was written in 2004 and had a focus on Sun Solaris 8. Since then Linux-based systems became more and more popular, therefore we will also spend some words about (Red Hat/Fedora) Linux.
- During this course we will learn basic commands whose syntax and options are mostly identical in all Unix flavours. Differences will be pointed out (but be aware that the author might not have found all differences…).
- For interactive use we concentrate on the Bourne Again Shell (bash).
- Shell prompt in command examples: `$`
- Commands and terminal output are written in `Courier font`. Optional input is surrounded by `[ ]`. Placeholders for e.g. file names are written in *`italic font`*. When a new command is introduced the first time, it is written `coloured`.

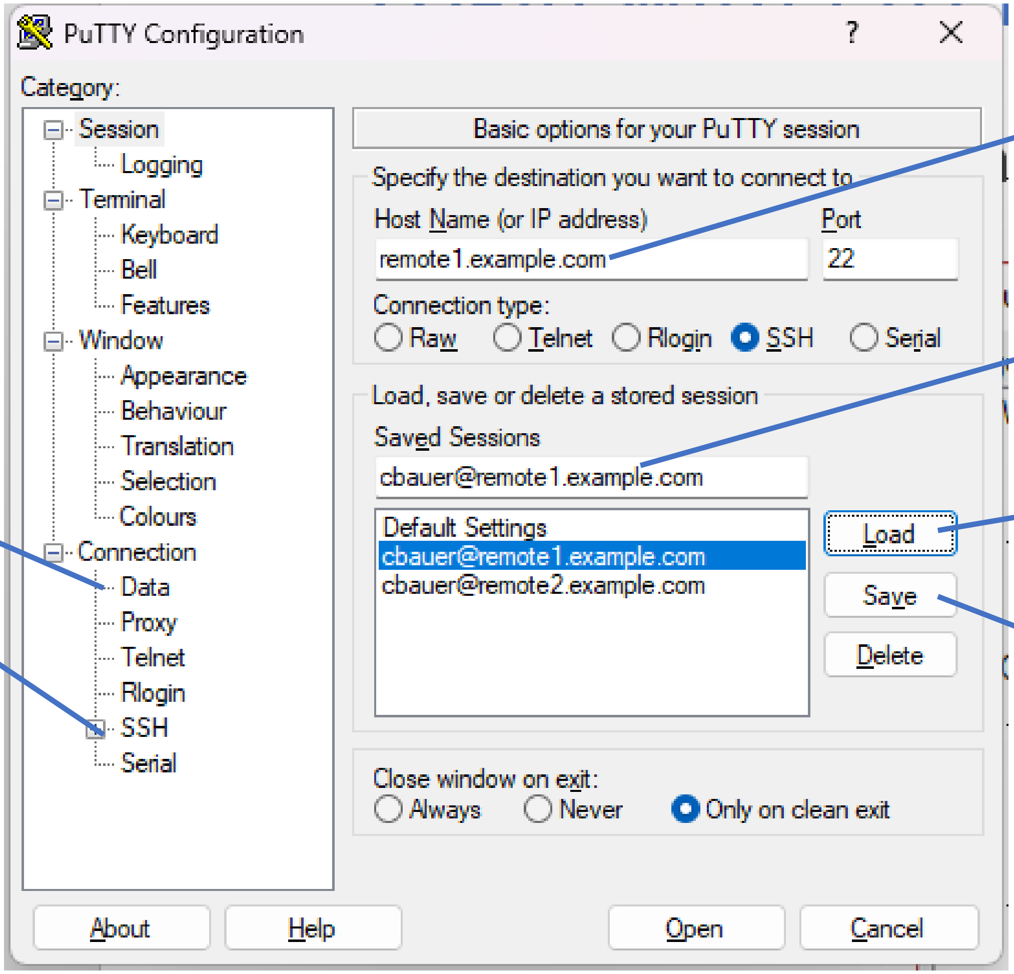
Architecture

- Unix is an interactive, time-sharing operating system (OS).
- Core of the OS: the kernel
  The kernel manages the utilisation of hardware resources (CPU, memory, I/O devices, physical storage, ...) by all running programs.
- Every piece of software running on the OS (except most of the kernel systems) is represented as a process.
- Some running processes waiting for actions to take are called "daemons."
  Daemons lie dormant until certain events occur that trigger an action.
- Interfaces between users and system: shells (these are yet other processes)
- A graphical user interface (GUI) is not considered being part of the OS.
  GUIs are made up of a server (the "X server") managing the I/O resources (mouse, keyboard and graphical output devices) and clients (programs using the graphical capabilities of the system).

Login with PuTTY

- PuTTY application: "Session" window

Further settings here, to make life easier…



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): remote1.example.com

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions: cbauer@remote1.example.com

Default Settings
cbauer@remote1.example.com
cbauer@remote2.example.com

Close window on exit: Always Never [✓ Only on clean exit]

Host name (must be resolvable by your PC) or IP address

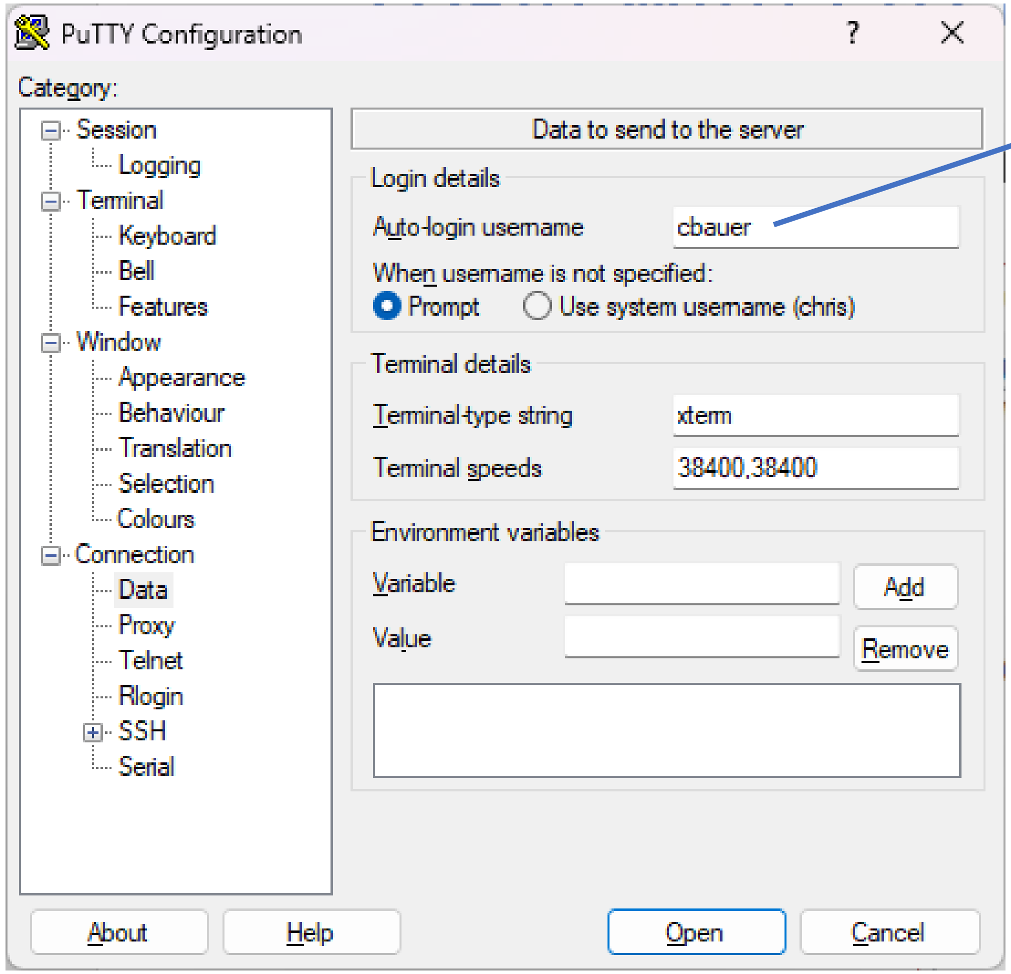
Define a name for the PuTTY session

Load stored session (attention: save your changes first!)

Save session, settings will be stored under the name entered above

Login with PuTTY

- PuTTY application: "Connection" – "Data"



Data to send to the server

Login details

Auto-login username: cbauer

When username is not specified:
[✓ Prompt] Use system username (chris)

Terminal details

Terminal-type string: xterm

Terminal speeds: 38400,38400

Environment variables

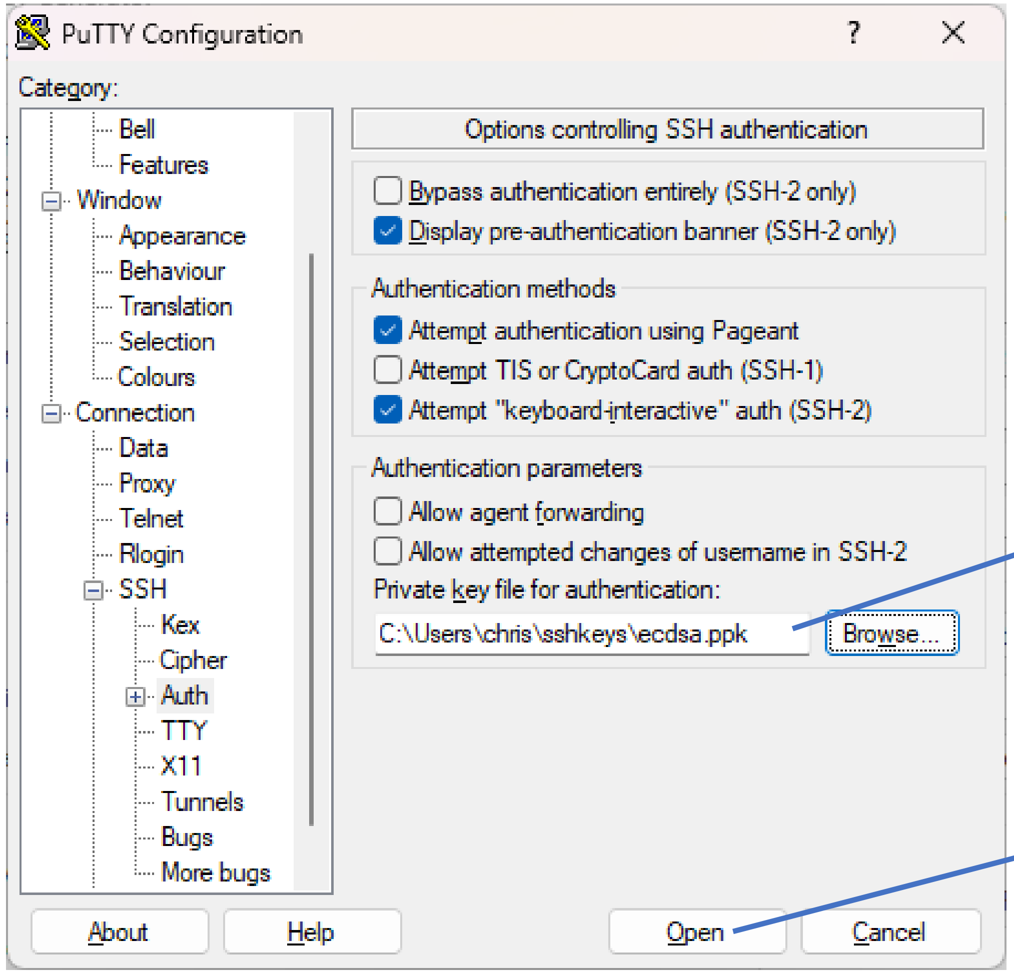
Variable:

Value:

Enter user name which shall be automatically logged in

Login with PuTTY

- PuTTY application: "Connection" – "SSH" – "Auth"


PuTTY Configuration

Category:

- Bell
- Features
- Window
  - Appearance
  - Behaviour
  - Translation
  - Selection
  - Colours
- Connection
  - Data
  - Proxy
  - Telnet
  - Rlogin
  - SSH
    - Kex
    - Cipher
    - Auth
    - TTY
    - X11
    - Tunnels
    - Bugs
    - More bugs

Options controlling SSH authentication

- [ ] Bypass authentication entirely (SSH-2 only)
- [✓] Display pre-authentication banner (SSH-2 only)

Authentication methods

- [✓] Attempt authentication using Pageant
- [ ] Attempt TIS or CryptoCard auth (SSH-1)
- [✓] Attempt "keyboard-interactive" auth (SSH-2)

Authentication parameters

- [ ] Allow agent forwarding
- [ ] Allow attempted changes of username in SSH-2

Private key file for authentication: C:\Users\chris\sshkeys\ecdsa.ppk

Choose private SSH key from file on disk

Press "Open" after you have made (and saved) all settings

Login with PuTTY

Local PuTTY configuration on Windows PC

- The PuTTY settings are stored in the Windows Registry.
- Export PuTTY settings to a file on the Desktop: enter
  ```
  regedit /e "%userprofile%\desktop\putty-registry.reg"
  HKEY_CURRENT_USER\Software\Simontatham
  ```
  in the command window or Windows Powershell.
- Import the settings on another PC: right-click the generated `.reg` file and choose "Merge" ("Zusammenführen") from the menu.

Use the tool PuTTYGen for key generation and management and conversion of key pairs generated with `ssh-keygen` to PuTTY key format (ppk) and vice versa.

Accessing a Unix system

- You can use `ssh` to log in to another system at the command line:
  `ssh [-l` *`newuser`*`] [-X]` *`hostname`*
- Users are authenticated via user name and password either stored in `/etc/passwd` and `/etc/shadow` or in an IAM (Identity and Access Management) system's `passwd` database (e.g. LDAP).
- The login shell is read from `passwd`:

$$\underbrace{\texttt{cbauer}}_{\text{User name}}:\underset{\text{x means password is stored in shadow file}}{\texttt{x}}:\underbrace{\texttt{43277}}_{\text{UID}}:\underbrace{\texttt{30403}}_{\text{GID}}:\underbrace{\texttt{Bauer;Christoph…}}_{\text{GECOS field (comment, can contain any char-acter except ':')}}:\underbrace{\texttt{/home/cbauer}}_{\text{Home directory}}:\underbrace{\texttt{/bin/ksh}}_{\text{Login shell}}$$

  The login shell reads its standard initialisation files for setting the environment from the user's home directory.
- Log out with the `exit` command.
- `–X` option: X11 forwarding

Accessing a Unix system

- The `ssh` login can be configured such that no password is required. A public/private key pair is required. Generate one on command line:
  `ssh-keygen -t` *`type`* `[-f` *`outputfile`*`]`
- You can define a "passphrase" which, if set, must be entered instead of the password. It provides additional security.
- Key type can, for example, be `rsa` or `dsa` (encryption method).
- By default, RSA keys are stored in `$HOME/.ssh/id_rsa[.pub]`, DSA keys in `$HOME/.ssh/id_dsa[.pub]`. Be careful not to overwrite existing key files! (`ssh-keygen` asks for the output file name if not specified with `-f`.)
- On the target host (to which you want to log in using ssh), the public key must be included in the `$HOME/.ssh/authorized_keys` file.

```
$ cd ~/.ssh
$ cat id_rsa.pub >> authorized_keys
```

Accessing a Unix system

- What is the operating system of the host I am logged in on?

```
uname -a
```

- Output for Solaris 10 system

```
SunOS host01 5.10 Generic_150400-55 sun4v sparc sun4v
```

- Output for Red Hat Linux system

```
Linux host02 2.6.32-696.13.2.el6.x86_64 #1 SMP Fri Sep 22
12:32:14 EDT 2017 x86_64 x86_64 x86_64 GNU/Linux
```

- On Solaris systems, the OS release can be found in `/etc/release`, on Linux systems you can query `/proc/version` or `/etc/os-release`.

Accessing a Unix system

- The `xterm` command (example) opens a new terminal window:

```
xterm -ls -sb -sl 2000 -bg antiquewhite -fg black
-title "cbauer host01" -n "cbauer host01"
-geometry 130x40+20+20 &
```

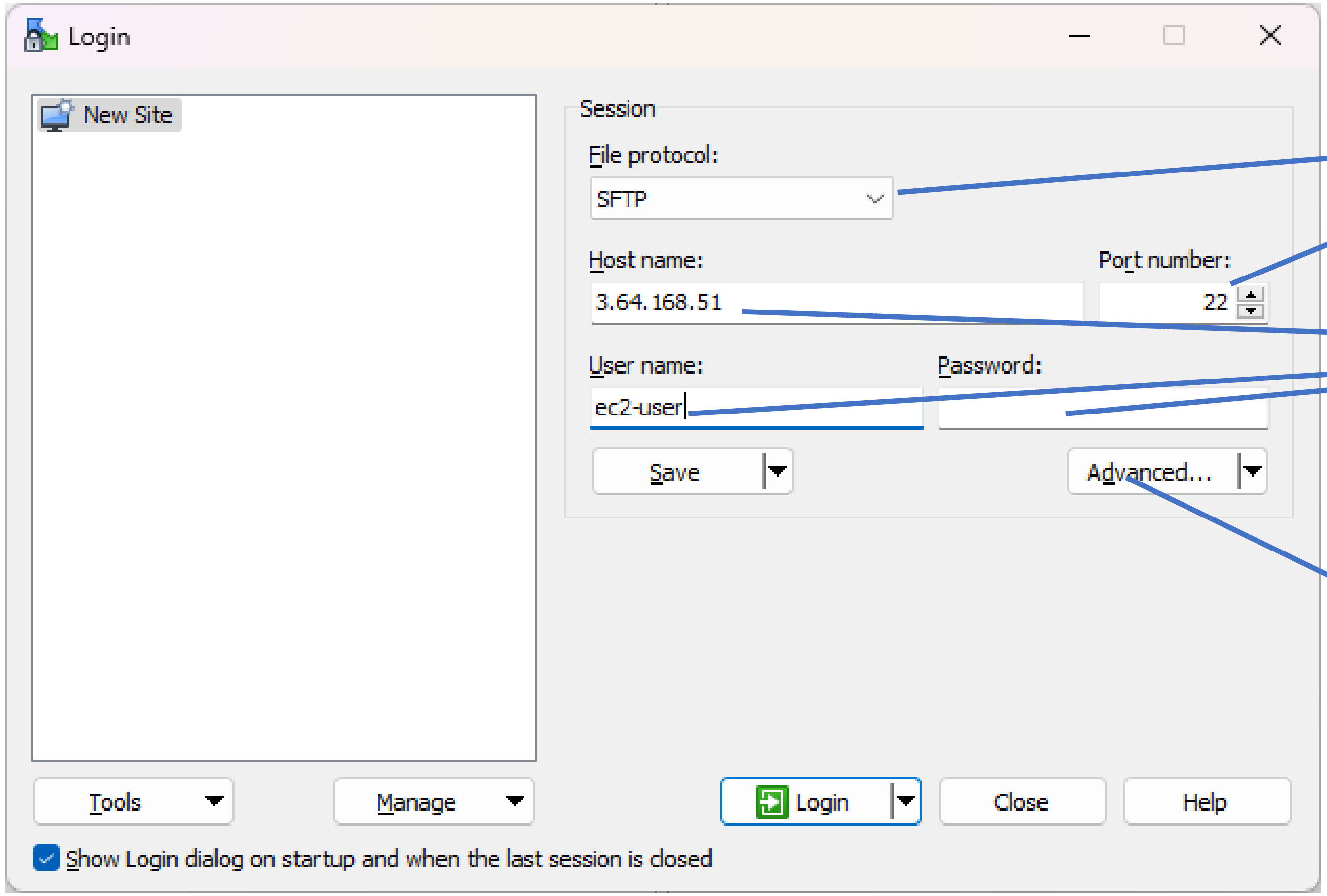
- Options used in the example above:

`-ls`	shell that is started in the window is a login shell
`-sb`	display scrollbar and save lines scrolled off the top
`-sl ...`	save this number of lines scrolled off the top (def.: 64)
`-bg ...`	background colour (https://en.wikipedia.org/wiki/X11_color_names)
`-fg ...`	foreground colour
`-title ...`	title string displayed in the window border
`-n ...`	name string displayed in the icon when minimised
`-geometry <`*`cols`*`>x<`*`rows`*`>+`*`x`*`+`*`y`*	

- Typical location is `/usr/bin` (Red Hat) or `/usr/openwin/bin` (Solaris).
- On your host/PC, an X Windows Server must be installed and running, e.g. XMing (Windows) or one of the Unix/Linux X Servers such as Gnome or KDE.

Exchanging data between Windows PC and Unix hosts

- WinSCP is a graphical frontend for Secure FTP (SFTP).


Login

New Site

Session

File protocol: SFTP

Host name: 3.64.168.51

Port number: 22

User name: ec2-user

Password:

[✓] Show Login dialog on startup and when the last session is closed

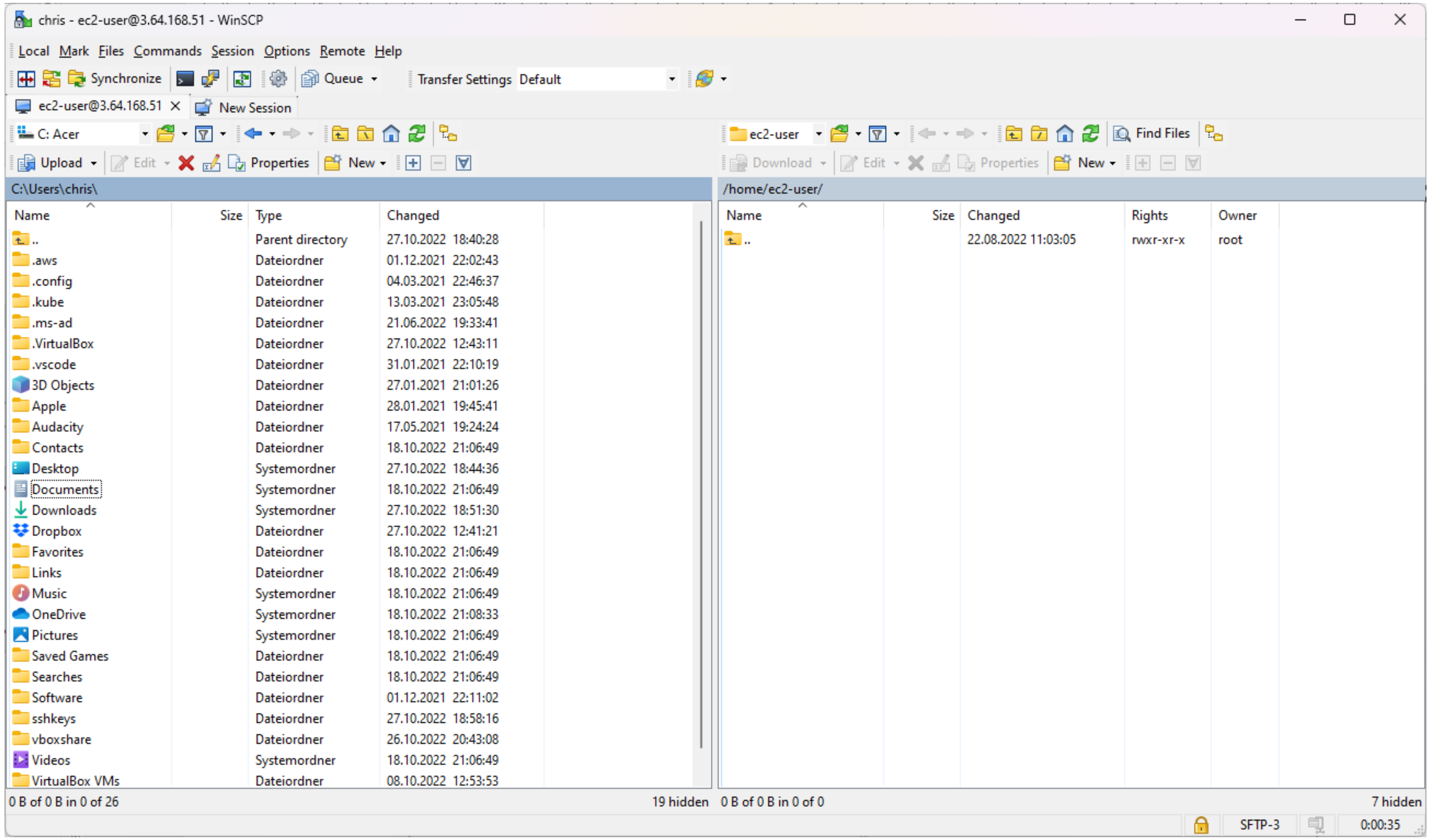
Choose "SFTP" protocol and port 22

Enter all data required for the login: user name, password and remote host name

If you're using an SSH key, configure it here ("SSH" – "Authentication")

Exchanging data between Windows PC and Unix hosts

- Files can be dragged and dropped between PC (left) and remote Unix host (right).



C:\Users\chris\

Name	Size	Type	Changed
..		Parent directory	27.10.2022 18:40:28
.aws		Dateiordner	01.12.2021 22:02:43
.config		Dateiordner	04.03.2021 22:46:37
.kube		Dateiordner	13.03.2021 23:05:48
.ms-ad		Dateiordner	21.06.2022 19:33:41
.VirtualBox		Dateiordner	27.10.2022 12:43:11
.vscode		Dateiordner	31.01.2021 22:10:19
3D Objects		Dateiordner	27.01.2021 21:01:26
Apple		Dateiordner	28.01.2021 19:45:41
Audacity		Dateiordner	17.05.2021 19:24:24
Contacts		Dateiordner	18.10.2022 21:06:49
Desktop		Systemordner	27.10.2022 18:44:36
Documents		Systemordner	18.10.2022 21:06:49
Downloads		Systemordner	27.10.2022 18:51:30
Dropbox		Dateiordner	27.10.2022 12:41:21
Favorites		Dateiordner	18.10.2022 21:06:49
Links		Dateiordner	18.10.2022 21:06:49
Music		Systemordner	18.10.2022 21:06:49
OneDrive		Systemordner	18.10.2022 21:08:33
Pictures		Systemordner	18.10.2022 21:06:49
Saved Games		Dateiordner	18.10.2022 21:06:49
Searches		Dateiordner	18.10.2022 21:06:49
Software		Dateiordner	01.12.2021 22:11:02
sshkeys		Dateiordner	27.10.2022 18:58:16
vboxshare		Dateiordner	26.10.2022 20:43:08
Videos		Systemordner	18.10.2022 21:06:49
VirtualBox VMs		Dateiordner	08.10.2022 12:53:53

0 B of 0 B in 0 of 26 19 hidden

/home/ec2-user/

Name	Size	Changed	Rights	Owner
..		22.08.2022 11:03:05	rwxr-xr-x	root

0 B of 0 B in 0 of 0 7 hidden

SFTP-3 0:00:35

Shells

You can always invoke another shell than your login shell; choose one of:

- Bourne Shell (`sh`) – the oldest and simplest shell with the least features, but available almost everywhere and still widely used in system administration
- Korn Shell (`ksh`) – an extension to the Bourne Shell
- Bourne Again Shell (`bash`) – another extension to the Bourne Shell

These shells are not available everywhere (depends on installed packages), but are worth mentioning:

- C Shell (`csh`) – a shell with C-style programming syntax (incompatible with Bourne and Korn Shell),
- Tenex C Shell (`tcsh`) – an extension to the C Shell,
- Z Shell (`zsh`) – often considered as being the shell with the most features.
- Almquist Shell (`ash`) – another extension of sh, lightweight, e.g. for embedded Linux systems

This course will concentrate on bash. Although they sound like, `rsh` ("remote shell") and `ssh` ("secure shell") are no shells.

User environment

- Immediately after login, you have a specific environment.
- Most settings are stored in initialisation files:
  - System-wide initialisation files, depending on your login shell
  - Personal initialisation files, depending on your login shell
- Settings can be: (environment) variables, aliases, functions, resource limitations, defaults for specific commands (e.g. `chmod`: the `umask` value) or shell-specific option settings.

Side note: almost everything in Unix is case-sensitive!

User environment

Variables store values which are referenced by names

- Some variables are automatically set by the shell (e.g. `PWD`), some are set by the system (e.g. `PATH`, `MANPATH`, ...).
- Use `=` to assign a value to a variable.
- Access to variable contents: `$`*VAR* or `${`*VAR*`}`
  E.g., `$HOME` is the contents of the variable `HOME`.
- Use the `echo` command to display contents of a variable.
- Example for variable usage:

```
$ PATH=/usr/bin:/usr/local/bin
$ echo $PATH
/usr/bin:/usr/local/bin
$ echo "The PATH is: $PATH"
The PATH is: /usr/bin:/usr/local/bin
$ PATH=$PATH:/opt/Acrobat5/bin
```

User environment

- Variable names may contain alphanumeric characters and _.
- Remove (unset) a variable: `unset` *`VAR`*
- Display all variables: `set`
- Set an environment variable: `export` *`VAR`*`[=`*`value`*`]`
  Environment ("global") variables are inherited by child processes and subshells.
- Display all environment variables:

  ```
  typeset -x
  export
  env
  ```

- Switch between env./local variable: `typeset ±x` *`VAR`*
  (`typeset -x` *`VAR`* is identical with `export` *`VAR`*)

User environment

The PATH variable

- Example: `PATH=/usr/bin:/usr/local/bin`
- The `PATH` variable tells the shell where to search for commands.
- The `PATH` directories are searched from left to right; the first executable found is used.
- Executable is not in the search PATH: use relative or absolute path name:

```
$ ./mycommand
$ /home/usera/mycommand
```

The Unix password

- The password can be modified using the `passwd` command.
  - If the password is stored in an external IAM system, it may be required to use its GUI.
- The encoded password is stored in `/etc/shadow` (not readable for non-root users) or the `shadow` IAM (e.g. LDAP) database; encoding is a one-way function.
  The encoding algorithm is the same for all Unix versions.
- Default (minimum) password rules for Solaris environments (maybe outdated)
  - all characters on the keyboard (except the Return key) may be used,
  - the password is case-sensitive (beware of pressed CAPS LOCK key!),
  - the password must be at least 6 characters long,
  - only the first 8 characters are significant (even though more may be typed),
  - within the first 6 characters there must be at least two letters and one special or numeric character.
- Other Unix flavours or your site-specific setup may enforce additional rules or periodic modification of the password ("password aging").

Getting help

Manual pages

```
man [-s section] [-M path] command
```

- Examples:

```
man ls
man -s 3head signal
```

- Search path for man pages: `MANPATH` variable
- Browse the man page with:

space	one page forward
b	one page backwards
[*n*]return	scroll forward *n* lines (default 1)
/*pattern*	search forward for *pattern*
n	search next occurence of *pattern*
q	quit

Getting help

Manual pages

- Sections (selected):

1	User commands and application programs
1m	System administration commands and daemons (Solaris)
2	System calls and error numbers
3	Functions and libraries
4	File formats (Solaris), devices and special files (Red Hat)
5	Miscellaneous, e.g. standards, env, macros (Solaris), file formats (Red Hat)
6	Games and demos
7	Special files (Solaris), miscellaneous (Red Hat)
8	System administration tools and daemons (Red Hat)
9	Device driver interfaces (Solaris)

- Sections may contain subsections, e.g. 3head or 3lib.
  Sections may be referred to, like in "see `termio(7i)`".

Getting help

Manual pages

- Introduction to a section:
  `man [-s` *`section`*`] intro`
- If more than one man page for the same topic (command or file name,...) is available, `man` will display the one which is found first by scanning the subdirectories of the `MANPATH` directories.
- Search for a topic/man page:
  `apropos` *`string`*
  `man -k` *`string`*
  (requires that `windex` files are available $\rightarrow$ contact your system administrator if you get error messages)
- Brief description only: `whatis` *`command`*

Unix commands

The general synopsis of using a command is:

```
command [options] [arguments]
```

Every command is

- an executable file located in the directory tree (binary, script),
- a shell-builtin function,
- a user-defined shell function or
- a shell alias name for another command.

A command returns an (integer) exit status to the parent command; by convention, 0 means "success", >0 (up to 255) some error. Use `echo $?` in order to check the latest return code.

Unix commands

- Executable files are searched in the directories listed in the `PATH` variable; the first executable file found matching the command name is executed.
- Files in directories that are not in `PATH` must be called with their path name (absolute, e.g. `/usr/ucb/whoami`, or relative, e.g. `../ucb/whoami`). An executable or script in the current directory can only be executed if either `.` is part of the `PATH` variable or `./` is prepended.
- Use the `which` command to see which binary executable is used. Example:

```
$ which cd
/usr/bin/cd
```

`which` does not show whether an alias or a shell-builtin with that name exist - these would take precedence over the binary executable! The Korn Shell has a built-in command `whence` combining the functionality of `which` with alias and shell-builtin information. Or you use the `type` command, available in ksh and bash.

Unix commands

- Options normally have a leading – character, as in
  `ls -a -l -i -s`
- Several options can be combined with one leading –, as in
  `ls -lisa`
- Some options may require their own arguments, as in
  `ssh -l cbauer remote1`
- Arguments are passed to the command as additional information input; beware that the shell may change argument values if the arguments contain shell metacharacters (e.g. `*`).

Basic commands

- Display who I am: `id [-a]`, `who am i` or `whoami` (on Solaris `whoami` is located in `/usr/ucb`.)
- Display the host I am on: `hostname` or `uname -a`
- Display the current date and time: `date`
- Display how long a command execution takes: `time` , as in e.g. `time date`
- Display a calendar: `cal [ [`*`month`*`] [`*`year`*`] ]`
- Display who else is logged on: `who`
- Display who was logged on at what time: `last [`*`username`*`]`
- Display the current session's pseudo terminal: `tty`
- Clear the screen: `clear`
- Go to sleep: `sleep` *`secs`*
- A simple calculator: `bc` (`CTRL-d` to exit)

Basic commands

- Use the `ls` command to list the contents of directories.
  Common options:

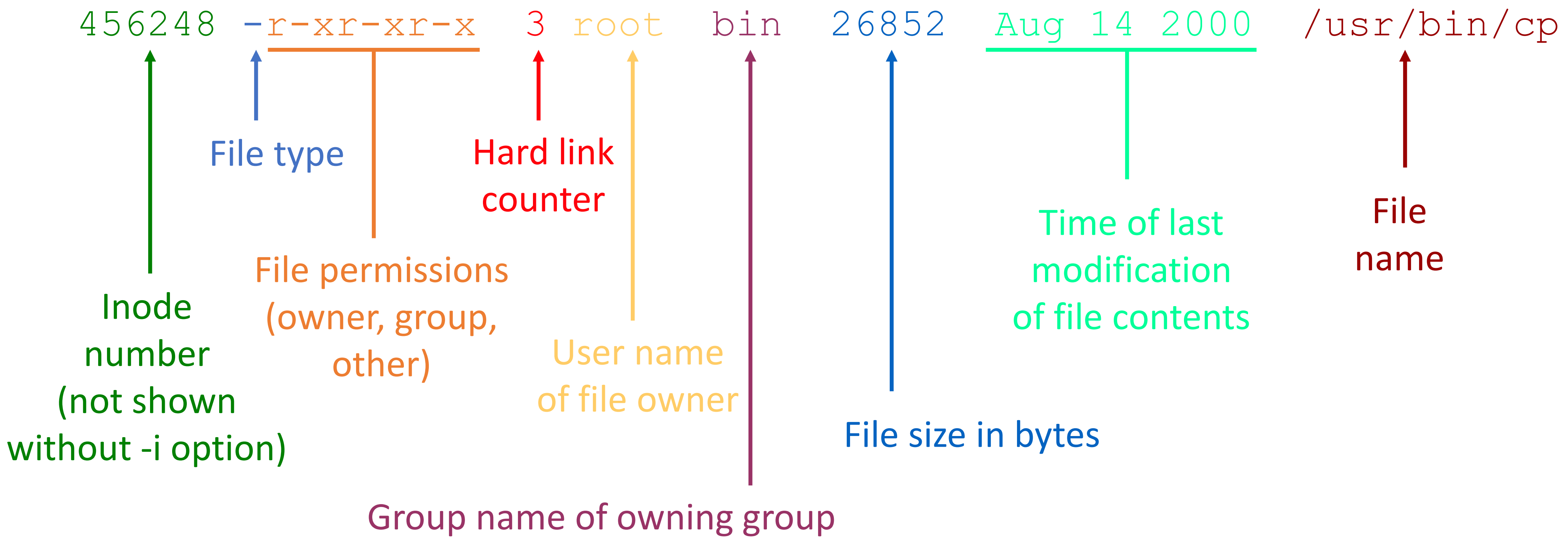
`-a`	list all entries, including those starting with `.` ("hidden files")
`-l`	long format
`-R`	recursively list subdirectories
`-d`	if argument is a directory, list its name, not its contents (often used with `-l`)
`-F`	mark special files with special characters (e.g. directories with a trailing `/`)
`-i`	print inode number for all entries
`-n`	same as `-l`, except that UID and GID are given instead of user/group name
`-s`	for every entry, print size in blocks[1]
`-t`	sort by time stamp instead of by name (default last modification time)
`-c`	use time of last inode modification for sorting (`-t`) and printing (`-l`)
`-u`	use time of last access for sorting (`-t`) and printing (`-l`)
`-r`	reverse the order of sorting

[1]The block size depends on the type of file system used and its method of allocating space on the disk

Basic commands

- Example:

```
$ ls -li /usr/bin/cp
456248 -r-xr-xr-x  3 root  bin  26852  Aug 14 2000  /usr/bin/cp
```


Inode number (not shown without -i option)
File type
File permissions (owner, group, other)
Hard link counter
User name of file owner
Group name of owning group
File size in bytes
Time of last modification of file contents
File name

Files in Unix

- Logically: a stream of data with a beginning and an end ("end of file" EOF, represented by CTRL+d)
- Physically: a set of disk blocks (sectors) in a file system (on a disk)
- Every file has an inode associated with it which stores:
  - owner and owning group, access permissions,
  - file type (directories are merely a special type of file!) and size,
  - number of hard links to the file (file names),
  - time stamps: last modification, last access, last modification of inode,
  - locations of data blocks on the physical storage.
- There is no concept of "extension-based" file types like in Windows, at least on terminal/shell level.
- Files and directories starting with `.` are not displayed by `ls` by default, only if the option `-a` is given: "hidden files".

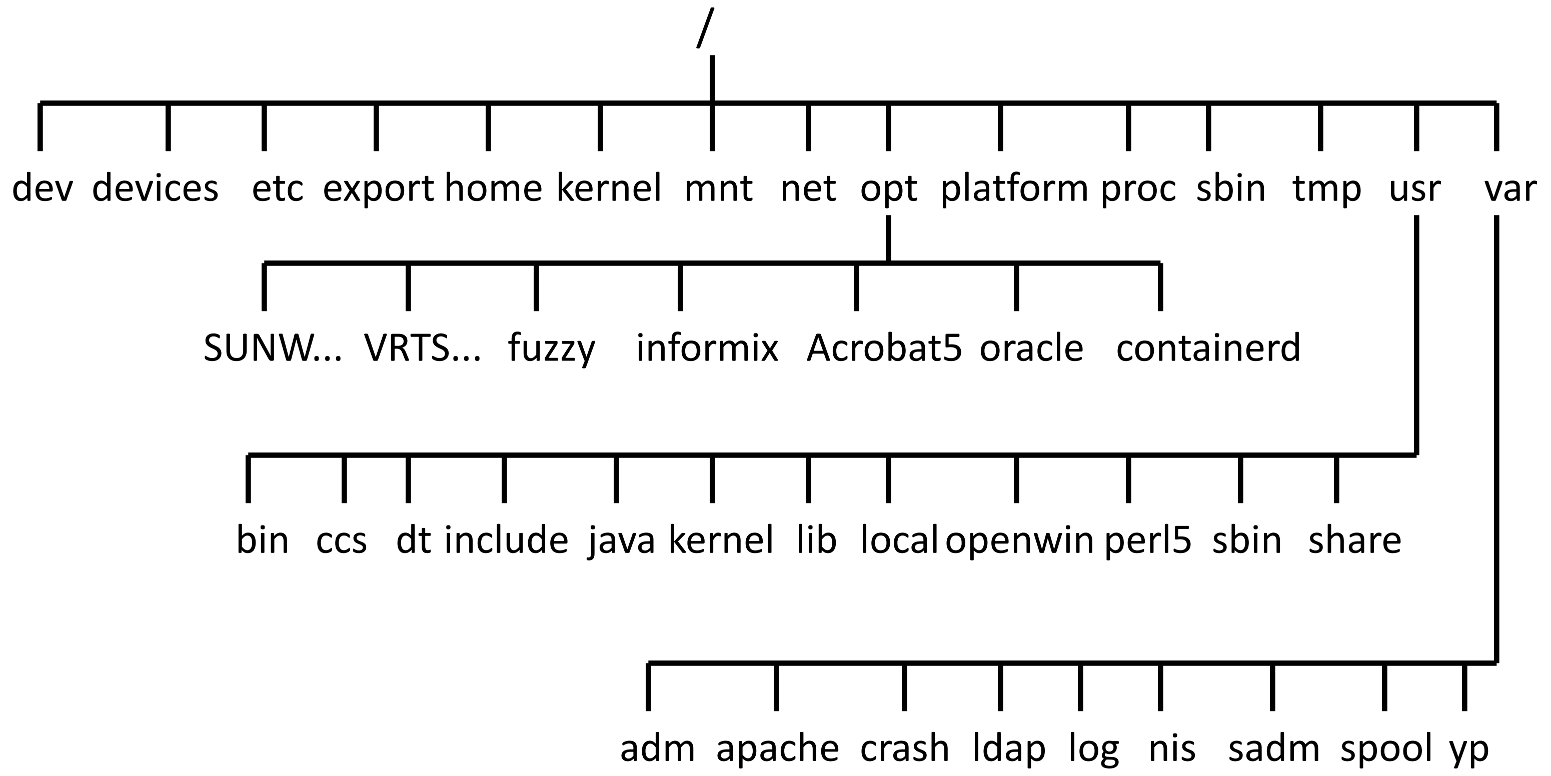
Hard and symbolic links

- Hard links: several names for one physical file (no special file type)
  `ls -l` command: the link counter indicates the number of hard links.
  All the names point to the same inode (physical address in file system).
- Soft or symbolic links: special files that point to other objects (similar to "links" = "Verknüpfungen" in Windows)
- Create a hard [or symbolic] link with the `ln` command:
  `ln [-s]` *`orig_file link_file`*
- Example:

```
$ ln testfile testfile2
$ ln -s testfile testfile3
$ ls -li
 466616 -rw-r--r--   2 cbauer   cbauer        94 Jan 19 08:35 testfile
 466616 -rw-r--r--   2 cbauer   cbauer        94 Jan 19 08:35 testfile2
 466556 lrwxrwxrwx   1 cbauer   cbauer         8 Jan 19 08:36 testfile3 -> testfile
$ rm testfile
```

Directory structure

- All files are organized in a single coherent directory tree. Typical Sun Solaris tree:


/
dev devices etc export home kernel mnt net opt platform proc sbin tmp usr var
SUNW... VRTS... fuzzy informix Acrobat5 oracle containerd
bin ccs dt include java kernel lib local openwin perl5 sbin share
adm apache crash ldap log nis sadm spool yp

Directory structure

- Directories are merely lists of file (and directory) names.
- Absolute path names:

```
/opt/oracle/product/11201_cl_64
/opt
/
```

- Relative path names:

```
product
./product/11201_cl_64
```

- Every directory may contain an arbitrary number of subdirectories and files.
- `.` is a reference to the current directory, as in `./product`
- `..` is a reference to the parent directory, as in `../../opt`
- Every directory contains `.` and `..`
  (for the `/` directory `..` is a reference to itself)

Directory structure

- Change directory using the `cd` command: `cd [`*`dir`*`]`

  ```
  cd /opt/oracle/product
  cd ../..
  ```

- `cd` without arguments takes you to your home directory (`$HOME` is the standard argument to `cd`).
- `cd .` takes you nowhere. You stay where you are.
- Determine the current directory with the `pwd` command ("print working directory").
- The shell provides a `PWD` variable; read its contents, e.g., with `echo $PWD`.

Important standard directories

- `bin` is a standard subdirectory for executables, as in `/usr/bin`, `/usr/local/bin`, …
- `lib` is a standard subdirectory for shared libraries (and other shared data), as in `/usr/lib`, `/usr/local/lib`, …
- `man` is a standard subdirectory for man pages, as in `/usr/share/man`.
- `include` is a standard subdirectory for header files (`*.h`), as in `/usr/include`.
- `/dev` and (Solaris) `/devices` contain device special files for I/O to devices.
- `/etc` contains system-wide configuration files.
- `/export` (Solaris) is a standard directory for things to be exported to other systems, e.g. via NFS (home directories etc.).
- `/root` is the exclusive home directory of the root user.

Important standard directories

- `/home` is a standard system directory for local or NFS-mounted home dirs.
- `/kernel` and `/platform` (Solaris) or `/boot` and `/lib/modules` (Red Hat) contain the OS kernel and kernel modules.
- `/bin` is a link to `/usr/bin` in Solaris, on Red Hat `/bin` contains basic commands required at all system run levels.
- `/lost+found` is a standard directory where the Unix file system check (`fsck`) puts lost data found during the check.
- `/mnt` is a standard system directory where file systems (e.g. NFS) are mounted.
- `/net` is a standard system directory for "browsing" NFS servers.
- `/opt` ("optional") is a standard directory for "optional" software packages, often used for commercial third-party software.
- `/proc` is a virtual (in-memory) storage for process structures and details, used by programs such as `ps`

Important standard directories

- `/sbin` is for system administration commands and daemons that must be available during system boot.
- `/tmp` is a storage for temporary data, volatile (in virtual memory) on Solaris, on disk on Red Hat.
- `/usr` ("Unix system resource") contains static software and data.

`/usr/bin`	binaries for user commands
`/usr/sbin`	binaries for system administration commands
`/usr/dt`	(Solaris) components of the Common Desktop Environment
`/usr/openwin`	(Solaris) X11 binaries and Open Windows components
`/usr/include`	header files
`/usr/lib`	shared libraries
`/usr/share`	shared data (man pages, libraries, ...)
`/usr/ucb`	(Solaris) Berkeley (BSD) compatibility stuff
`/usr/ccs`	(Solaris) some things used in SW development

Important standard directories

- `/var` ("variable") is for non-static data (logs, protocols, system accounting, mails, print spooler files, on-disk temporary data).

`/var/adm`	system administration data (message files, logs, ...)
`/var/log`	log data
`/var/mail`	incoming mails for local Unix users (outgoing go through `/var/spool/mqueue`)
`/var/spool`	"spooled" data (print jobs, mails, software packages, ...)
`/var/run`	storage space for the OS, in Solaris this is volatile (in memory)
`/var/tmp`	on-disk (non-volatile) temporary storage for user data

Manipulating files and directories

Copy a file or directory

`cp [opts] source[s] destination`

- No. of args > 2: destination must be a directory, all source objects are copied to there.
- Common options:

`-r`	recursive, copy a directory and all its contents
`-R`	same as `-r`, except that pipes are replicated, not read from
`-p`	preserve owner and group, permissions and time stamps
`-i`	interactive: ask before overwriting a target (not default!)

- Copying directories requires use of the `-r` or `-R` option.

  If the *`destination`* directory already exists, a subdirectory will be created within there whose name is *`source`*.

Manipulating files and directories

Move (rename) a file or directory

```
mv [opts] source[s] destination
```

- No. of args > 2: destination must be a directory, all source objects are moved to there.
- Directories move with all their contents (no "recursive" option).
- Common option:

  `-i` interactive: ask before overwriting a target (not default!)

Manipulating files and directories

Delete (remove) a file or directory

`rm [opts] object[s]`

- Objects are normally regular files, although `rm` can also remove symbolic links, files of other types, and (with the recursive option) even directories.
- Common options:

`-i`	interactive: ask before overwriting (not default!), recommended
`-r`	recursive (use with care and caution); required for directories
`-f`	remove write-protected files without prompting

- There is no undo for a remove operation! There is no "wastebasket!" A backup must be restored in case of accidental removal.

Manipulating files and directories

Create an empty file or update time stamps

`touch [opts] object[s]`

- If the objects do not exist, `touch` will create empty files.
- If an object already exists, the access and modification time stamps will be updated with the current system time.
- Common options:

`-a`	change the access time of the file only
`-m`	change the modification time of the file only
`-c`	do not create a file if it does not exist
`-r` *`file`*	use the time stamps of *`file`* instead of the current time
`-t [[CC]YY]MMDDhhmm[.ss]`	set explicit time stamp

Manipulating files and directories

Create a new empty directory

`mkdir [opts] directory`

- Use the `-p` option to create a full path, e.g. `mydir1/mydir2`.

Remove an empty directory

`rmdir [opts] directory`

- Use the `-p` option to remove a directory and its parent directories (directories must recursively become empty in order for `rmdir` to be successful, meaning that there must not be any remaining files in subdirectories).

Displaying files

Display file type

`file` *`file[s]`*

Display file contents

`cat` *`[opts] file[s]`*

- Normally used for text files (ASCII data files, log files, scripts etc.)
- Common options

`-n`	precede each output line with its line number
`-v`	print non-printable characters visibly (see man page)
`-ve`	print `$` char at the end of each line (Red Hat: `-e`)
`-vt`	print tabs as `^I` and form-feeds as `^L` (Red Hat: `-t`)
`-u`	unbuffered output (for fast copies; Solaris only)

Displaying files

- Redirect output to another file: for example
  `$ cat file1 file2 > file3`
  (can be used to combine several files into one, e.g. after splitting a file with the `split` command; works with text or binary files)

Display printable character sequences in a (binary) file

`strings` *`file`*

(sometimes useful to obtain information about the contents, purpose or nature of a binary file)

Displaying files

Display file contents

```
more [opts] file[s]
```

- Normally used for text files (ASCII data files, log files, scripts etc.)
- Most often used without options
  (options see man page)
- Keys for scrolling and searching:

space	one page forward
b	one page backwards
[*n*]return	scroll forward *n* lines (default 1)
/*pattern*	search forward for *pattern*
n	search next occurence of *pattern*
q	quit

Displaying files

Display file contents

`less [opts] file[s]`

- Normally used for text files (ASCII data files, log files, scripts etc.)
- Most often used without options
  (options see man page)
- Keys for scrolling and searching: similar to `man` and `more`
- `less` is similar in functionality to `more`, but `less` knows a lot more interactive key commands than `more` (see man page); e.g. `?` for searching backwards.
- With the `PAGER` variable, `/usr/bin/less` may be defined as alternative pager command for the `man` command.

Displaying files

Display the beginning or the end of a file

```
head -n file[s]
tail ±n file[s]
```

- *n* indicates a number of lines to display; default 10.
  `−`*n* display the first/last *n* lines
  `+`*n* display from the *n*th line up to the end (`tail` only; Solaris only)
- `tail` is often used on log files in the form
  `tail -f` *file*
  ("follow mode"): `tail` does not exit after displaying the current end of the file but waits for additional lines to be appended to the file and displays them in an instant.

Displaying files

View binary files: "octal dump"

`od [opts] file`

- Useful for binary files for which `cat` or `more` merely display rubbish
- Common options

`-a` or `-t a`	interpret bytes as named characters
`-b` or `-t o1`	interpret bytes in octal
`-c` or `-t c`	display single-byte characters (non-printable chars as 3-digit octal numbers)
`-x` or `-t x2`	display words (2-byte units) in hex

(more formatting options see man page)

Displaying files

Count bytes, words (delimited by white space/newline) and lines in files

`wc [opts] file[s]`

- Common options

`-l`	display line count only
`-c`	display byte count only (including newlines!)
`-w`	display word count only

`wc` without options displays line, word and byte count.

Comparing files

`diff [opts] file1 file2`

- Compares two ASCII text files
- Common options

`-b`	ignore trailing white space (spaces and tabs) and treat other white space sequences as equivalent
`-i`	ignore case of letters
`-w`	ignore all white space (spaces and tabs)
`-r`	recursive, compares all files in directories (Red Hat)

Comparing files

`sdiff [opts] file1 file2`

- Compares two ASCII text files, with one on each half of the screen; differences are indicated by >, < and | in the middle between the files.
- Common option

`-s`	ignore identical lines

`cmp [opts] file1 file2`

- Compare files byte by byte, can be used for ASCII and binary files.
- Display all differences (decimal byte number, octal differing bytes): `cmp -l`

Comparing directories

`dircmp [opts] dir1 dir2`

- Compares the contents of two directories file by file
- Common option

  `-s` do not list identical files in the report
- `dircmp` is only available on Solaris; both on Solaris and Red Hat `diff -r` serves a similar purpose.

Shell basics (bash)

The prompt variable

- The shell prompt can be customized.
- Assign the shell variable `PS1`.
- bash provides a set of specific control sequences for prompts.
- Example:
  ```
  PS1="\u@\h:\w \d \t \nEnter command: "
  ```
  yields the prompt:
  ```
  cbauer@host01:/etc Thu Mar 25 16:40:04
  Enter command:
  ```

Shell basics (bash)

Keys for command line editing when using the shell interactively:

- Cursor Left and Right: move through the line
- Cursor Up and Down: scroll up and down through command history
- CTRL+a, CTRL+e: jump to beginning/end of line
- ESC+f, ESC+b: jump forwards/backwards word by word
- CTRL+d or "Entf" key: delete character at cursor position
- Backspace key: delete character before cursor position
- CTRL+l: clear the screen (keeping the current command line)
- CTRL+r, CTRL+s: search history backwards/forwards (terminal CTRL+s must be turned off)
- Tab key: automatic command and file completion
- CTRL+c: interrupt command without execution and return to prompt

More command line editing key combinations exist. Note that bash must be in "emacs mode" (`set -o emacs`) when you want to use the key combinations above, alternative would be `set -o vi` for `vi`-style command line editing. The same options apply for the Korn Shell.

Shell basics (bash)

Shell options are parameters that control the behaviour of the shell.

- Switch on a shell option: `set -o` *`option`*
- Switch off a shell option: `set +o` *`option`*
- Display all current option settings: `set -o`
- bash options are, for example:

`noclobber`	Overwrite protection on > redirection
`ignoreeof`	Shell does not exit on CTRL+d
`emacs`	emacs-style command line editing (with cursor keys)
`noglob`	Turn off expansion of shell wildcards (`*` etc.)

Shell basics (bash)

Initialisation files

- Initialisation files permanently store variable definitions, shell option settings, aliases, prompt variables etc.
- Login bash shells read:
  `/etc/profile` - system-wide
  `$HOME/.bash_profile`, `$HOME/.bash_login`, `$HOME/.profile`
  (the first one of these three that exists and is readable is used).
- Interactive bash shells which are not login shells read:
  `$HOME/.bashrc`.
- "Source" a specific profile file interactively or from another file:
  `.` *`filename`*
  (requires leading `./` when `.` is not in the `PATH` variable; `source` can be used, too)

Shell basics (bash)

Input/output redirection

- By default, processes take input from keyboard (stdin, file descriptor 0) and write output and errors to the screen (stdout, 1; stderr, 2).
- Redirect input (read from file): `<`

  ```
  mailx -s "Subject" "christoph.bauer@vodafone.de" < message.txt
  ```

- Redirect output (write to file): `>` (overwrite) or `>>` (append)

  ```
  ls -lR > outfile
  ls -lR >> outfile
  ```

- Redirect error output (write to file): `2>`

  ```
  ls -lR 2> /dev/null
  ```

  `/dev/null` is a pseudo device that discards the output if it is of no interest.
- Redirect output and errors:

  ```
  grep cbauer file1 >/dev/null 2>&1
  ```

Shell basics (bash)

Input/output redirection

- When using >, files are overwritten (by default)!
  Switch on overwrite protection: `set -o noclobber`
  Switch off: `set +o noclobber`
  Bypass protection: use >| instead of >
- Pipeline (or "nameless pipe"): combine two processes such that output of first process is input to second process

```
ps -ef | grep cbauer
cat message.txt | mailx -s "Subject" "christoph.bauer@vodafone.de"
ps -ef | tee outfile | grep cbauer
```

The `tee` command writes the input from the first pipe into the file and, at the same time, passes it on into the next pipe. Hence it works like a "T junction."

Shell basics (bash)

Pipelines, lists and groups

- Pipeline: singular command or sequence of commands separated by `|`
- List: sequence of one or more pipelines separated by

`;`	strictly sequential execution
`&`	asynchronous execution (shell does not wait for a pipeline to finish before starting next)
`&&`	next list is only executed if preceding list returns exit code 0
`||`	next list is only executed if preceding list does not return 0

- `{`*`list`*`}` simply execute list (used to group commands)

  Example: `{ls .; ls ..; cat file1} > outfile`
- `(`*`list`*`)` execute list in a separate environment (used to group commands with modified effective settings)

  Example: `(cd subdir; ls -lR)`

Shell basics (bash)

Shell metacharacters

- Before executing a command, the shell tries to substitute all path and file names that contain metacharacters.
- Important metacharacters for file name substitutions:

`*`	any sequence of arbitrary characters (including none)
`?`	exactly one arbitrary character
`[`*`chars`*`]`	exactly one of these chars (range: e.g. `A-Z` or `0-9`)
`[!`*`chars`*`]`	exactly one char, but not one of the specified
`\`	masks the following character (loses its special meaning)

- Directory name metacharacters (tilde and dash expansion):

`~`	your home directory
`~`*`user`*	home directory of *`user`*
`-`	the last working directory (before the last `cd`)

Shell basics (bash)

Quoting and command substitution characters

- Sometimes character sequences must be quoted so that the shell does not interpret special characters.
- `'   '` mask all special chars except `'`
- `"   "` mask all special chars except `"  \  ` $`

  `'` and `"` mask each other.
- `\` masks a single char (including newline: mask end-of-line).
- `` `   ` `` command substitution: the command is executed and its output is inserted (substituted)

  Example: ``PATH=$PATH:`pwd` ``
- `$(   )` command substitution (alternative)

Shell basics (bash)

Aliases

- Aliases are useful shortcuts for often used commands
- `alias [name[='cmd [options]']]`
- Show [all] aliases: `alias [name]`
- `unalias name`
- Aliases are not exported to sub shells (use `.bashrc`).
- Use the original command if it has been overwritten by an alias with the same name: prepend it with `\`, e.g. `\rm -rf mydir`.

Searching text in files

Commands: `grep`, `egrep`, `fgrep`

`[e,f]grep [opts] 'searchstring' file[s]`

- "grep" stands for "Global Regular Expression Parser" (or "Globally search for a Regular Expression and Print if found").
- `grep` is the regular form of `grep`.
- `egrep` means "expression grep" (an alternative to `grep`).
  `egrep` may be faster, but also more memory-consuming. In Red Hat, `egrep` is linked to `grep` and functionally identical with `grep -E`. `egrep` uses a different regular expression set.
- `fgrep` means "fixed grep".
  `fgrep` can only search for fixed character sequences, it does not perform any regular expression pattern matching. Therefore it is faster than `grep` and `egrep`. In Red Hat, `fgrep` is linked to `grep` and functionally identical with `grep -F`.
- Quoting (`'  '` or `"  "`) of the search string is recommended.
  Otherwise the shell may interpret some of the RE metacharacters.

Searching text in files

- Simple examples of using `grep`:

```
grep 'Bauer' /etc/passwd
grep '[Bb]auer' /etc/passwd
grep '[Bb]...r' /etc/passwd
grep '[Bb].*r' /etc/passwd
grep '^[Bb]...r$' /etc/passwd
grep '^[Bb]...r\$$' /etc/passwd
grep '[A-Z][a-z]..r' /etc/passwd
```

Searching text in files

- Common options of `grep`:

`-n`	show line numbers
`-i`	ignore case of letters
`-v`	show all lines that do not contain the search pattern
`-l`	show only names of files that contain the search pattern
`-c`	count the lines that contain the pattern

Regular expressions

- Combinations of ASCII characters, some of which have special meanings
- Important special characters (metacharacters):

Symbols	Meaning
.	exactly one arbitrary character
*	preceding character arbitrarily often (including none)
^	beginning of line
$	end of line
\	masks the one following character
[-]	exactly one from this list (range) of characters
[^ -]	exactly one character, but not from this list (range)

Within [ ] the special characters . * [and \ lose their meaning.

Searching for files

`find dir[s] [condition[s]] [action[s]]`

- `dir[s]`: where to search (by default: including subdirectories)
- `condition[s]`: a set of conditions that must be satisfied
- `action[s]`: actions that are performed for all objects found
- If no condition is given, all objects will match.
- The default action is to list path names of matching objects.
- Actions are merely special conditions, i.e. an action yields a Boolean value, too, which depends on the return value of the command.
- For a comprehensive list of all conditions and actions see man page.

Searching for files

`find dir[s] [condition[s]] [action[s]]`

- Conditions: true if...

`-name pat`	object's name is *pat* (shell patterns allowed)
`-atime n`	object was last accessed *n* days ago
`-ctime n`	object's inode was changed *n* days ago
`-mtime n`	object was last modified *n* days ago
`-newer file`	object was modified more recently than *file*
`-inum n`	object's inode number is *n*
`-links n`	object has *n* links
`-user name`	object is owned by user *name*
`-group name`	object is owned by group *name*

Searching for files

`find` *`dir`*`[`*`s`*`] [`*`condition`*`[`*`s`*`]] [`*`action`*`[`*`s`*`]]`

- Conditions: true if...

`-perm [-]`*`on`*	object's perm. flags match octal number *`on`*

  (with the preceding −, only those bits that are set in *`on`* are compared with flags)

`-size `*`n`*`[c]`	object's size is *`n`* blocks (512 byte; `c`: bytes)
`-type `*`t`*	object's type is *`t`* (one of `f`, `d`, `b`, `c`, `l`, `p`, `s`)

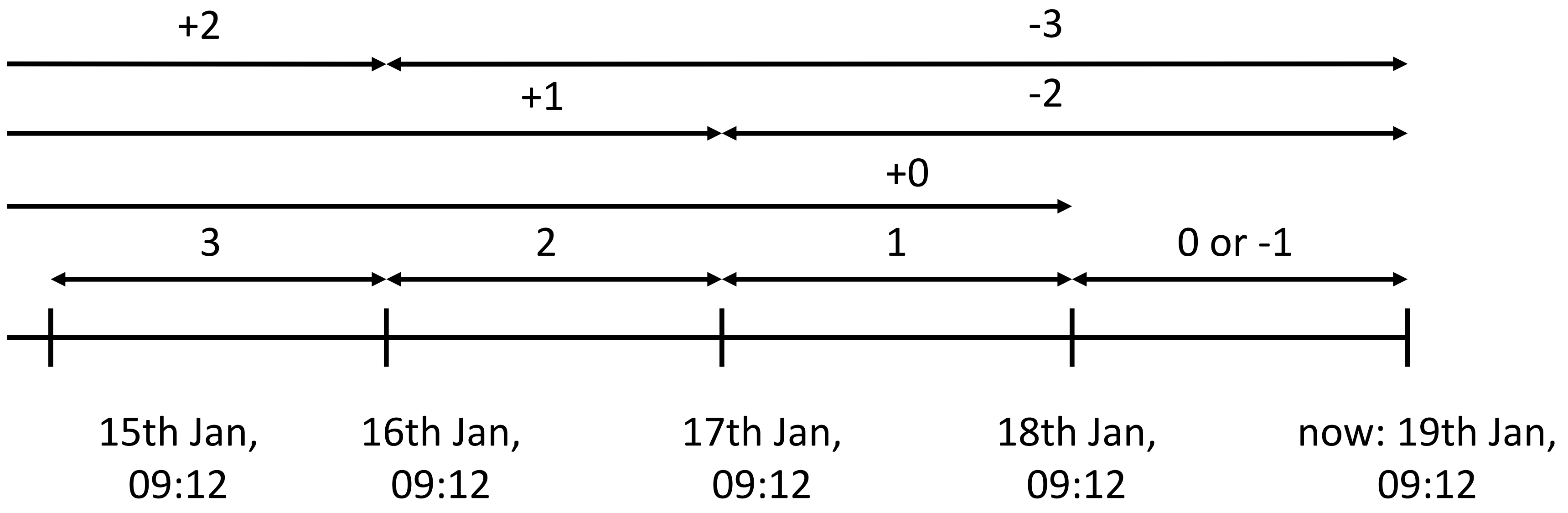
- Conditions can be combined with `-a` (AND) or `-o` (OR).
  Logical expressions can be parenthesized (`\( \)`).
  `-a` is the default, simply listing conditions means AND implicitly.
  `!` means a logical NOT.

- Numerical values:
  `+`*`n`* means "more than *`n`*", `-`*`n`* means "less than *`n`*".

Searching for files

`find dir[s] [condition[s]] [action[s]]`

- What does `mtime ±n` mean?
  Consider 24-hour periods, take no notice of the midnight break between days.



Searching for files

`find` *`dir`*`[`*`s`*`] [`*`condition`*`[`*`s`*`]] [`*`action`*`[`*`s`*`]]`

- Actions

`-print`	simple path name output (default action)
`-ls`	detailed output, format like `ls -lid`
`-exec` *`cmd`* `\;`	execute *`cmd`* for every object found; command arg `{}` is replaced by current object.
`-ok` *`cmd`* `\;`	like `-exec`, but user confirmation required

`-exec` and `-ok` yield true when the exit status of *`cmd`* is 0.

- Example:

```
find /home/cbauer -type f -name '*core*' -exec rm {} \; -ls
```

(Red Hat Linux: `-ls` must be before `-exec rm`, error message otherwise)

Note: before executing such a command, it is always good to check which files would be affected, by running `find` without the `-exec` option.

Editors

Several editors are available in Unix:

- `ed`, `ex` are single-line editors (similar to `edlin` in MS-DOS)
  `ex` is a superset of `ed`.
- `vi` is a terminal-fullscreen ("visual") editor based on the functionalities of `ex`; it is the standard Unix editor everyone should be able to use.
- `nano` is another simple text editor which became popular in the Linux world.

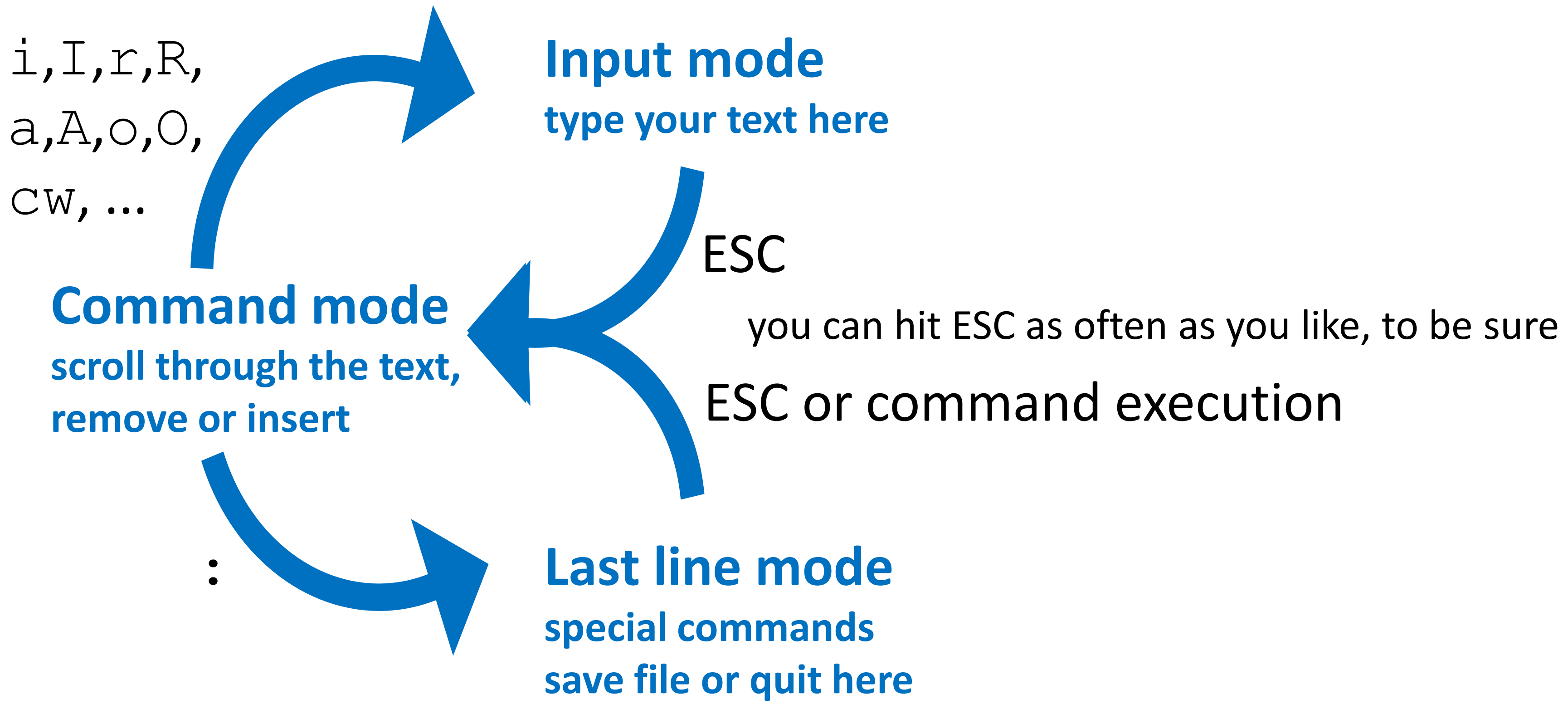
These editors are not included in Unix (Solaris) by default:

- `vim` ("vi improved") is an enhanced `vi`. On Linux, `vim` is usually started when `vi` is entered.
- `emacs` is another display-based editor; runs as a terminal-window based version or within its own graphical window; a variant delivered with some programming IDEs is `xemacs`.
- `dtpad` and `nedit` are examples of graphical editors.

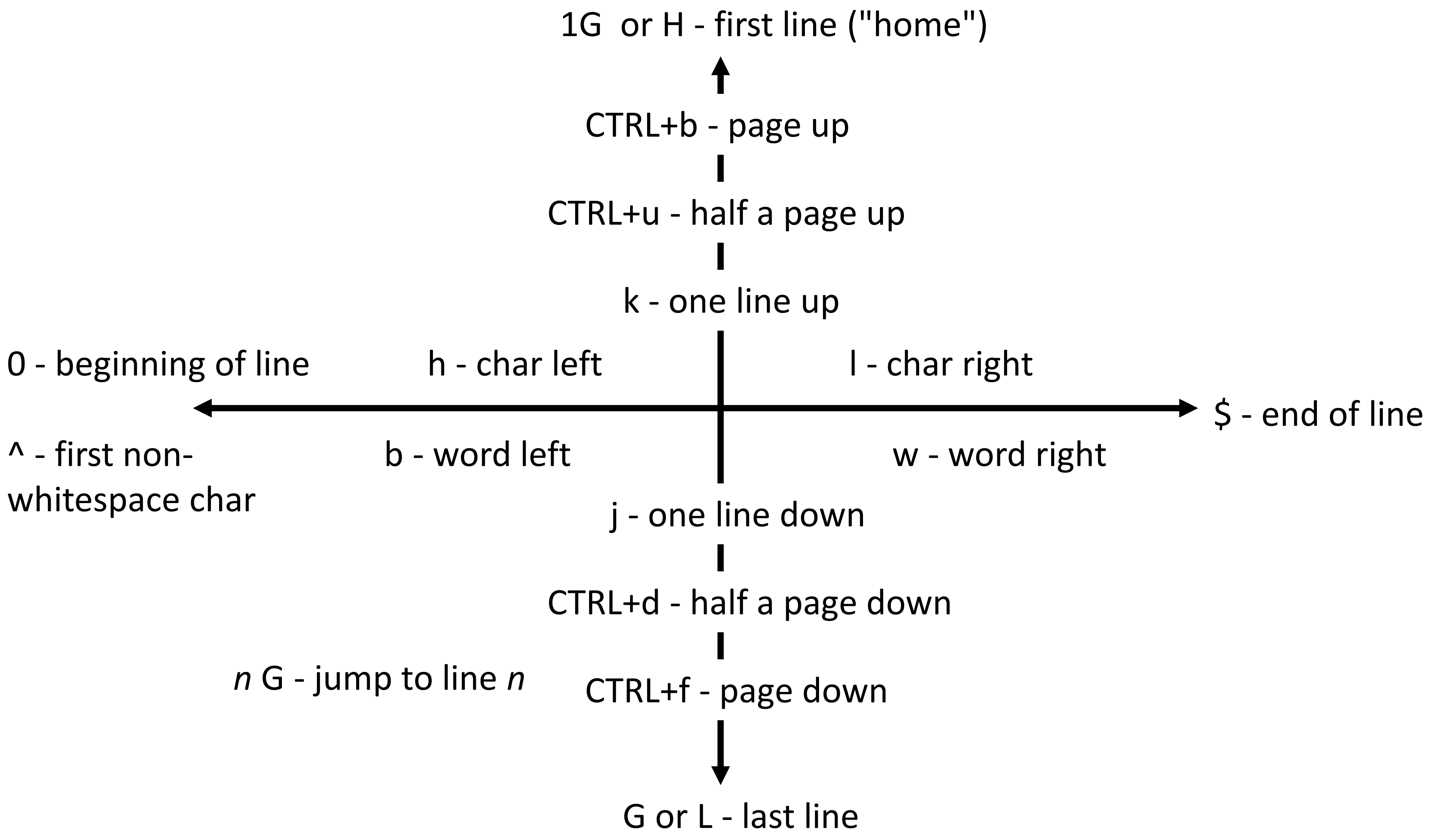
The `vi` editor

Startup: `vi [file]`

Three modes of operation



The `vi` editor



The `vi` editor

Entering input mode from command mode

`i`	enter input mode at current cursor position
`I`	enter input mode at beginning of current line
`a`	enter append mode after current cursor position
`A`	enter append mode after end of current line
`o`	open new line after current one and enter input mode there
`O`	open new line before current one and enter input mode there
`r`	replace one single character
`R`	enter replace mode, starting with current cursor position
`cw`	replace chars from current cursor position up to end of word

Except with `r`, you have to type ESC to return to command mode.

The `vi` editor

Command mode commands (cont.)

When typing a preceding number *n*, a command is repeated *n* times.

`dd`	delete current line to general buffer
`dw`	delete word (from cursor position) to general buffer
`D`	delete to end of line (from cursor)
`x`	delete single character (at cursor)
`p`	paste general buffer after cursor (current line)
`P`	paste general buffer before cursor (current line)
`J`	join current and next line into one line

The `vi` editor

Command mode commands (cont.)

When typing a preceding number *n*, a command is repeated *n* times.

`Y`	yank current line to general buffer
`yw`	yank word (from cursor position) to general buffer
`~`	toggle lower-/uppercase character by character
`.`	repeat last change
`u`	undo last change
`U`	undo all changes on current line
`CTRL+g`	display current line number and file information
`CTRL+l`	redraw screen
`ZZ`	save changes and exit (as `:wq`, see below)

The `vi` editor

Last line mode commands

`:w`	write current editor contents to current file
`:w` *`name`*	write current editor contents to file *`name`* (does not change the current file to *`name`*!)
`:wq` or `ZZ` or `:x`	write to current file and quit
`:q!`	quit without saving
`:!`*`cmd`*	execute shell command *`cmd`*
`:r` *`file`*	read *`file`* and insert its contents after current line
`:r !`*`cmd`*	execute shell command *`cmd`* and put output after current line
`:e` *`file`*	load file as new editor contents to edit

The `vi` editor

Last line mode commands

`:`*m*`,`*n*`t`*x*	copy lines *m* to *n* after line *x*
`:`*m*`,`*n*`m`*x*	move lines *m* to *n* after line *x*
`:`*m*`,`*n*`d`	delete lines from *m* to *n*

The `vi` editor

Search text

`/text`	search for `text` or regular expr. (forward direction)
`?text`	search for `text` or regular expr. (backward direction)
`n`	search next
`N`	search next in reverse direction

Search and replace: general syntax

```
    :[address]s/old_text/new_text/[g]
```

`address`	indicates the range of lines
`old_text`	text (or regular expression) to be replaced
`new_text`	text to be inserted instead

The `vi` editor

Search and replace

Line addressing:

- Explicit line number or `[g]`/*`pattern`*/ (regular expression match; `g` means all lines containing the pattern)
- `[`*`addr1`*`]`,`[`*`addr2`*`]` indicates a range of lines (first or last omitted: current line automatically inserted), `.` is the current line, `$` is the last line and `%` addresses all lines in the document.
- An appended `g` means that all occurrences of the text on a line are replaced (otherwise only the first occurrence would be considered).
- *`new_text`* may start with `&`: append *`new_text`* to *`old_text`*.

The `vi` editor

Search and replace: examples

```
:2,5s/cb/CB/
```

→ on line 2 to 5, replace the first cb with CB

```
:5,$s/cb/CB/g
```

→ on all lines from the 5th to the last, replace all cb with CB

```
:%s/CB/&AUER/g
```

→ in the entire document, append AUER to all CB

```
:g/CB/s/^123/456/
```

→ on all lines that contain the string CB, replace 123 at the beginning of the line with 456

The `vi` editor

Last line mode commands: editor settings

`:set number`	show line numbers
`:set autoindent`	autoindent after carriage return (`CTRL-d` to go back)
`:set showmode`	display current mode on last line of screen
`:set list`	show invisible characters
`:set tabstop=`*`n`*	define tabstop positions
`:set all`	show current values of all parameters

`:set` *`parameter`* is turned off with `:set` *`noparameter`*

Store your personal parameter config in `$HOME/.exrc`

File permissions

Viewing file permissions

```
$ ls -l testfile
-rw-r--r--   2 cbauer   cbauer  (...) testfile
```

The first block in this output is file type and permissions (user/owner, owner group and "others" = "rest of the world").

Files:	r	file contents can be read and displayed
	w	file contents can be modified
	x	file can be executed as a command
Directories:	r	dir contents can be read and displayed (`ls`)
	w	directory contents can be modified (creation and deletion of files and subdirectories)
	x	operation on the directory and its subdirectories is allowed

File permissions

- After authentication (login or switch UID), every access to an object is allowed or denied based on file permissions.
- Authentication is performed based on user names, access permissions are granted based on numerical UID/GID.
- The root user (and only the root user), UID 0, is allowed to do (almost) everything on the system, independent of access permissions!
  "With great power comes great responsibility."

File permissions

Modifying file permissions

`chmod [opts] permissions file[s]`

- Common option: `-R` for recursive (all files and subdirectories)
- Symbolic mode: `u` for user/owner, `g` for group, `o` for others, `a` for all, `r/w/x` for particular permissions, `+` or `-` to grant or remove a permission, `=` to set all permissions at once
  Examples:
  ```
  chmod u+rwx file1
  chmod g-w file2
  chmod g+x,o+rx dir1
  chmod a=rw file3
  ```
- You can only use `chmod` on your own files, only root can change all files' permissions.

File permissions

Modifying file permissions

```
chmod [opts] permissions file[s]
```

- Octal modes:
  1) Map each of `r`/`w`/`x` in a triple to a binary digit
  2) Set this binary digit to "1" for all granted permissions
  3) Translate the resulting three binary numbers to octal
- Examples:

```
chmod 755 dir1
chmod 644 file1
```

File permissions

Umask value

`umask octal_perms`

- Used to specify default permissions for new files and directories.
- `umask` identifies (in octal mode) which permissions are removed from the system default, which is

  for files: `rw-rw-rw-` (666),

  for directories: `rwxrwxrwx` (777).
- Examples:

  `umask 0`: new files `rw-rw-rw-`, new dirs `rwxrwxrwx`

  `umask 022`: new files `rw-r--r--`, new dirs `rwxr-xr-x`

  `umask 077`: new files `rw-------`, new dirs `rwx------`
- `umask` is normally set in initialisation files such as `.profile`.

File permissions

Change owner

```
chown [-R] user object[s]
```

Change owning group

```
chgrp [-R] group object[s]
```

- `chgrp` can only be used when the user is the owner of the object and also a member of the destination group. `chown` can only be used by root.
- `–R` for recursive mode (include all files and subdirectories)
- Determine (your) group memberships (primary, secondary) with

  ```
  groups [username]
  id -a [username]
  ```

File permissions

Switch UID

`[sudo] su [-] [username] [-c command]`

- Changes UID and GID (for login or, with `-c`, command execution)
- With the – argument, *`username`*'s profile is read and effective and the `su` session starts in *`username`*'s home directory; without this argument, the current directory and environment remain unchanged.
- *`command`* (optional) is executed within the `su` session.
- In many settings, `su -` is bound to `sudo` mechanism in order to switch to root user without having to type in the root password. Type `sudo -l` to check which commands have been allowed to you with `sudo` privileges.
- As root user, you can `su` to any other user without entering the password.
- Successful and unsuccessful attempts of using `su` are logged by the system. Site rules may forbid to use `su` to switch to another personal user identity!

Process and job control

Display processes

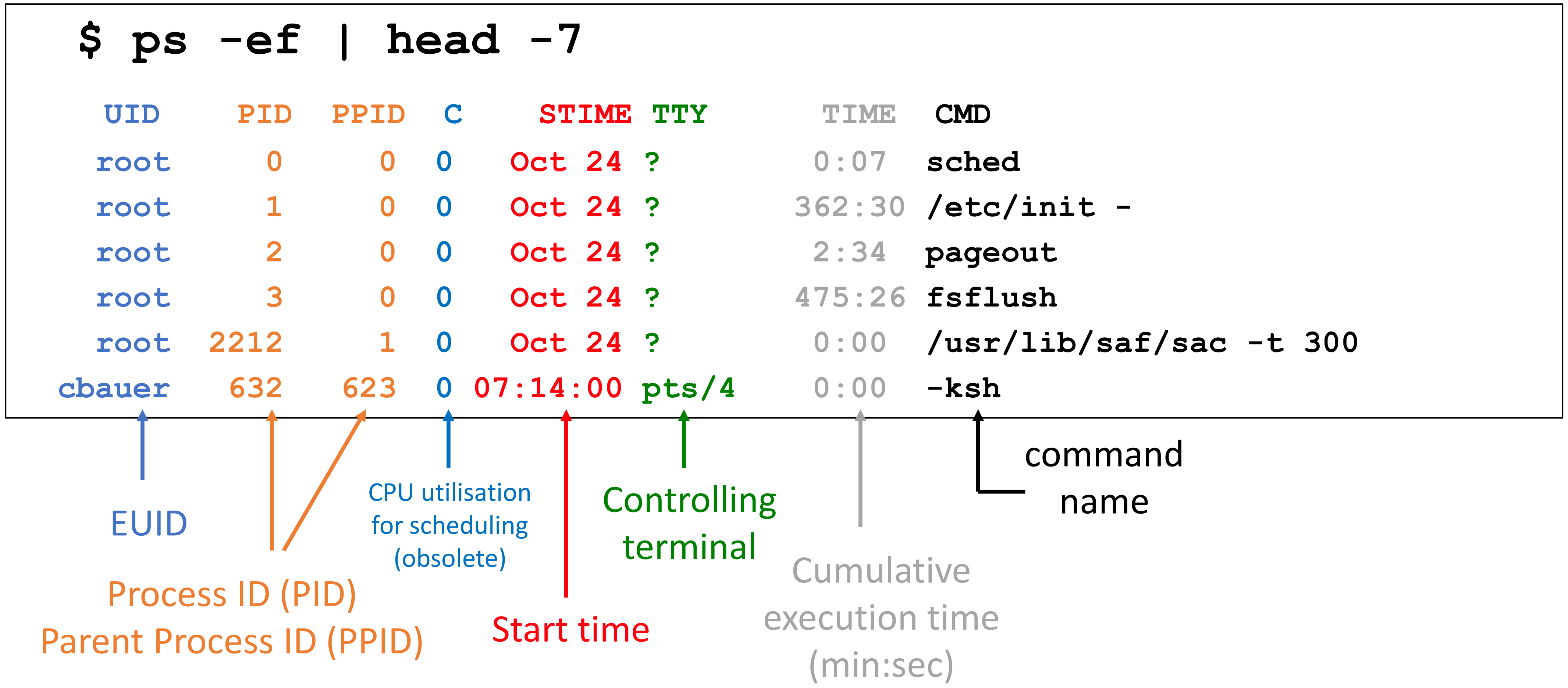
`ps [opts]`

- Without options, `ps` displays only processes that have the same effective UID and the same controlling terminal as the invoker of `ps`.
- `ps` generates a *snapshot* view of running processes.
- Common options:

`-e`	list information about every process now running
`-f`	generate a "full" listing (detail information)
`-l`	generate a "long" listing (detail information)

- `ps` can be used to display processes of specific groups (`-G`), process IDs (`-ps`), terminals (`-t`) or users (`-u`, `-U`) .

Process and job control



```
$ ps -ef | head -7

   UID     PID   PPID  C    STIME TTY       TIME  CMD
   root      0      0  0   Oct 24 ?         0:07  sched
   root      1      0  0   Oct 24 ?       362:30  /etc/init -
   root      2      0  0   Oct 24 ?         2:34  pageout
   root      3      0  0   Oct 24 ?       475:26  fsflush
   root   2212      1  0   Oct 24 ?         0:00  /usr/lib/saf/sac -t 300
 cbauer    632    623  0 07:14:00 pts/4     0:00  -ksh
```

Process and job control

Search for processes

```
ps [opts] | grep name [| grep -v grep]
```

```
pgrep [opts] name
```

- `pgrep` is available since Solaris 7 and works similar to `ps ... | grep ...`
- Common options to `pgrep`

`-l`	long output (not only PID, but also process name)
`-u` *user*	all processes for *user*

Process and job control

Signals

- Signals are notifications of events to running processes.
- Termination due to signal receipt: exit status = 128 + signal value
- A process may block or handle a signal appropriately.
- A process cannot block or handle SIGKILL and SIGSTOP.
- Generate signals to processes with the `kill` or `pkill` command:

  ```
  kill [-[s ]signal] PID[s]
  pkill [-signal] pattern
  ```

  - The default signal is 15 (SIGTERM) for both commands.
  - `pkill` reduces the awkward "`ps ... | grep ...`/pass output to kill" to one single command - use with care and caution!

Process and job control

Signals

- Important signals (see `signal(3head)` for Solaris or `signal(7)` for Red Hat):

Name	No	Def. Act.	Description
SIGHUP	1	Exit	Hangup
SIGINT	2	Exit	Interrupt (CTRL+c)
SIGQUIT	3	Core	Quit (CTRL+\)
SIGFPE	8	Core	Arithmetic Exception
SIGKILL	9	Exit	Kill
SIGBUS	10	Core	Bus Error (Solaris)
SIGSEGV	11	Core	Segmentation Fault
SIGTERM	15	Exit	Terminate
SIGUSR1	16	Exit	User Signal 1
SIGUSR2	17	Exit	User Signal 2
SIGSTOP	23	Stop	Stop (signal; CTRL+s)
SIGTSTP	24	Stop	Stop (user; CTRL+z)
SIGCONT	25	Ignore	Continue (CTRL+q)

Process and job control

Job control

- In Unix terminology a job is merely a command line of commands (may be several processes) invoked interactively from a shell.
- Job states: foreground, background or stopped
  (not to confuse with process states)
- By default jobs are running in the foreground (the shell is blocked).
- Start in the background: append an `&` sign, as in:
  `/usr/openwin/bin/xterm &`
  (the assigned job ID and the PID are displayed)
- Put a job into the background:
  1) Send a TSTP signal with `CTRL+z`
  2) Run the `bg` command

Process and job control

Job control

- The `jobs` command shows the current shell's jobs
- Refer to a job with `%`*`job_id`* (the number shown by `jobs`):

  ```
  kill %1
  bg %3
  ```

- Put a job to the foreground (default: last job that was put to `bg`)

  `fg [%`*`job_id`*`]`

- Stop a job: `kill -STOP %`*`job_id`*
- Reactivate a stopped job: `bg %`*`job_id`* or `fg %`*`job_id`*

Process and job control

Job control

- Background jobs terminate when the user exits from the invoking shell; the shell may prohibit the `exit` and notify the user of running or stopped jobs.
- Start a job so that it continues to run after exit:
  `nohup cmd &`
  (by default output goes to `nohup.out`; you may have to type `exit` twice to leave the shell)
  `nohup` tells the shell's children to ignore the SIGHUP signals sent by the exiting shell.
- Continuing background jobs are inherited by `init` (PID 1) when the shell exits.

Process and job control

Control-key sequences in terminals (see `termio(7i)` on Solaris, `termios(3)` on Red Hat)

- CTRL+c: interrupt (SIGINT), forces a process to terminate immediately
- CTRL+d: end of file (EOF), sometimes used to end an input stream
  CTRL+d exits a shell (use `set -o ignoreeof` in ksh and bash to deactivate this behaviour)
- CTRL+\: quit (SIGQUIT), forces a process to terminate immediately
  (same as CTRL+c except that CTRL+\ forces a core dump)
- CTRL+z: stop (SIGTSTP), stops a foreground process without blocking the terminal (shell)

Process and job control

Control-key sequences in terminals (see `termio(7i)` on Solaris, `termios(3)` on Red Hat)

- CTRL+s: stop (SIGSTOP), stops a foreground process and blocks the terminal (shell)
- CTRL+q: continue (SIGCONT), resume STOPped process
  (give it a try and type CTRL+q when your terminal appears to be frozen - you may inadvertently have typed CTRL+s)
- CTRL+v: special meaning of next control character is ignored
  (useful for input of control sequences in files – for example, for setting `stty erase`, see below)
- CTRL+h: backspace
  CTRL+?: delete
  Use `stty erase ^H` or `stty erase ^?` to define/change the erase key

Remote access

- Execution of a networking-related command requires a translation of the host name to an IP address.
  Check IP address translation:
  ```
  grep hostname /etc/hosts
  getent hosts hostname
  nslookup hostname[.domain]
  ```
- A network service requires a server process to run on the server host.
- Is the other host accessible via network (i.e. remote system is up and the network routing works)?
  ```
  ping [-s] remotehost
  ```
  On Red Hat, `ping` works as `ping -s` on Solaris. Note: some firewalls may restrict ICMP message exchange, hence `ping` might not work.
- Is a certain service available via network?
  ```
  telnet remotehost [port]
  ```

Insecure remote access

- `telnet` *`remotehost`*
  remote login session; telnet is an OS-independent standard.
- `ftp` *`remotehost`*
  file transfer (interactive session); FTP is an OS-independent standard.
- `rlogin [-l` *`other_user`*`]` *`remotehost`*
- `rcp [`*`remotehost`*`:]/`*`file`* `[`*`remotehost`*`:]/`*`file`*
- `rsh [-l` *`other_user`*`]` *`remotehost command`*
- The "r tools" are typical for Unix; their application, and that of `telnet` and `ftp`, is not recommended (and, often, not permitted anyway) due to security weaknesses. They transfer unencrypted traffic!

Secure remote access

- `ssh [-l` *`other_user`*`]` *`remotehost`* `[`*`command`*`]`
  encrypted remote login session or command execution; instead of `-l` *`other_user`* you may also type *`other_user`*`@`*`remotehost`*.
- `scp [[`*`other_user`*`@]`*`remotehost`*`:]/`*`file`*
  `[[`*`other_user`*`@]`*`remotehost`*`:]/`*`file`*
  encrypted remote file copy
- `sftp [`*`other_user`*`@]`*`remotehost`*
  encrypted file transfer (interactive, works similar to `ftp`)

Secure remote access

`sftp` example

```
$ sftp cbauer@remote1
Connecting to remote1...
sftp> ls
(...)
sftp> put testfile
Uploading testfile to /home/cbauer/testfile
sftp> get testfile2 testfile
Fetching /home/cbauer/testfile2 to testfile
sftp> quit
$
```

Archiving and compressing

`gzip` *`file`*`[`*`s`*`]`

`gunzip` *`file`*`.gz [`*`file`*`.gz ...]`

- `gzip` compresses file by file and appends `.gz` to every file.
- `gzcat` *`file`*`.gz` sends the file uncompressed to stdout (display) but leaves it compressed on disk. (Red Hat: use `gunzip -c` *`file`*`.gz` instead.)

`compress` *`file`*`[`*`s`*`]`

`uncompress` *`file`*`.Z [`*`file`*`.Z ...]`

- `compress` compresses file by file and appends `.Z` to every file.
- `zcat` *`file`*`.Z` sends the file uncompressed to stdout (display) but leaves it compressed on disk.

Archiving and compressing

```
zip [opts] file.zip file[s]
unzip [opts] file[.zip] [file[s]]
```

- `zip` compresses all files and stores the compressed data in one zip file.
- The ZIP format is compatible with PC ZIP formats (PKZIP, winzip, 7zip, ...).
- Common option to `zip`: `-r` for "recursive," used for directories
  Example: `zip -r home.zip ./*`
  It is recommended to archive using relative path names, in order to avoid access permission problems during extraction.
- Common option to `unzip`: `-l` for "list archived files" (no actual extraction)

Archiving and compressing

`tar [opts] [tarfile.tar] file[s]`

- One and the same command for archiving and extracting
- Common options:
  Action: one of `c` (create), `x` (extract), `t` (TOC=list), `u` (update), `r` (replace)

`v`	verbose
`f file`	name of the tarfile

- Examples

```
tar cvf home.tar .
tar tvf home.tar
tar xvf home
```
(extraction is always relative to current dir)

- It is recommended to archive using relative path names, in order to avoid access permission problems during extraction.

Finally: some useful commands worth noting

- Send HTTP(S) request to web server: `curl`
- Download a file via HTTP(S) or FTP from a web server: `wget`
- Mount file systems, for basic users: check mounted file systems: `mount`
- Total, used and available space of file systems: `df [-h] [`*`directory`*`]`
- Sum up file and directory sizes: `du [-sh] [`*`file(s)`*`]`
- Sync files and directories between hosts: `rsync`
- Trace system calls and signals: `truss` (Solaris) or `strace` (Red Hat)
- Check shared libraries which would be included (is my `LD_LIBRARY_PATH` variable set correctly?): `ldd`

See man pages to find out how to use these commands.

Finally: some useful commands worth noting

- Monitor running processes and their resource consumption: `top`
- Monitor system performance: `iostat`, `vmstat`, `sar`, `mpstat`, …
- Show network interfaces: `ifconfig [-a]`
- Show network connections, open ports, routing tables, …: `netstat [-arn]`
- Check and trace IP routing to remote destination: `traceroute`
- Query a DNS server: `nslookup`
- Copy files or devices byte by byte: `dd`

See man pages to find out how to use these commands.