Useful Shell Commands in Computational Linguistics

Max Kisselew
(based on slides from Antje Schweitzer)

Programming Course WS 2015 / 2016

2015-10-16

Note

This set of slides describes the most basic “every-day” shell commands. Some simple additional shell commands are introduced that are useful in many computational linguistics tasks (such as simple text statistics).

Covered Topics

1. The shell (3)
2. Navigating and manipulating the directory hierarchy (6)
3. The command line (5)
4. Copying, moving and deleting files (2)
5. Permissions (5)
6. Viewing files (2)
7. Standard in and standard out (5)
8. How to get information about commands (4)
9. head/tail, tr, sort, uniq, wc, tee (6)
10. Network (2)

What you should be able to do at the end of the day

- Know how to use the shell
- Know how to use the first important and useful Unix shell commands
- Know how to navigate through folders and how to view files without the graphical environment
- Know how to manipulate permissions in order to work in research groups
- Log in at the IMS servers from elsewhere

Overview

1. The shell
2. Navigating and manipulating the directory hierarchy
3. The command line
4. Copying, moving and deleting files
5. Permissions
6. Viewing files
7. Standard in and standard out
8. How to get information about commands
9. head/tail, tr, sort, uniq, wc, tee
10. Network

The shell

- is a program to interact with your operating system
- executes single commands
- or more complex programs consisting of many shell commands ("shell scripts")
- shell scripts are very helpful for automatizing typical sequences of commands
- there are different shells with slightly different syntax, the most wide-spread are probably **bash** (Bourne shell), **csh** (c-shell), **tcsh** (pronounce t-c-shell)
- **bash** is assumed throughout this tutorial

Check shell type

- `echo $SHELL`
  outputs your current shell type

- not necessary for most commands shown here but just in case:
- to get another shell (for example, the **tcsh** shell), type
  `tcsh`
- to always get a **tcsh** shell, change your configuration:
  `ypchsh`

The shell prompt

- the shell prompt indicates that the shell is ready to accept commands
- it can be configured to show the name of the computer and/or the current directory
- in addition, the prompts are often numbered

In Linux and Unix, case matters!
`file.wav` and `File.wav` and `FILE.WAV` are all different files and may even co-exist

Overview

1 The shell

2 Navigating and manipulating the directory hierarchy

3 The command line

4 Copying, moving and deleting files

5 Permissions

6 Viewing files

7 Standard in and standard out

8 How to get information about commands

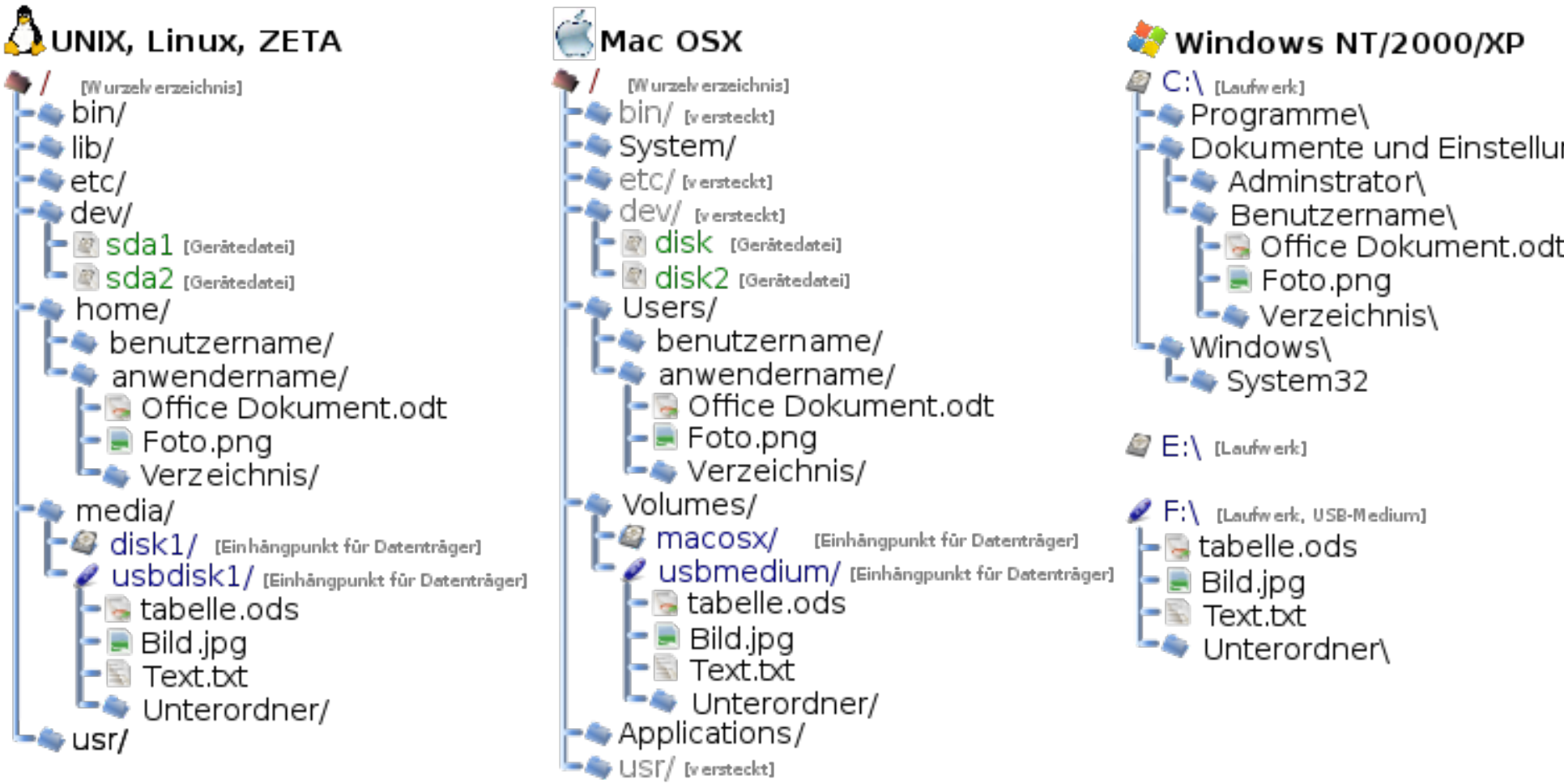
9 head/tail, tr, sort, uniq, wc, tee

10 Network

The directory hierarchy

- in Linux (and Unix) operating systems, all directories are organized in one directory tree (no drive letters)
- the symbol for the root directory is `/`
- paths along the hierarchy: directory names joined by `/`
  - `/home/users0/daniel/somedirectory` (absolute path)
  - `daniel/somedirectory` (relative path)
  - the previous paths refer to the same directory if current directory is `/home/users0`

Linux folder hierachy



(Source: `https://de.wikipedia.org/wiki/Verzeichnisstruktur`)

Most important directories

- `/home` directory contains users' home directories
- the directories below `/usr` contain files needed for various software (the programs themselves as well as source code, libraries etc.)
- `/mount/studenten` contains directories related to specific courses at IMS
- `/mount/projekte` contains directories for the project groups

Useful shortcuts for navigating the directory hierarchy

`/`	root directory
`~`	your own home
`~daniel`	home directory of user `daniel`
`..`	one directory up
`.`	the current directory

Examples

- `~peter/methodsCL/exercise1/solutions`
- `~/methodsCL/slides`
- `~/../daniel`
  (inefficient, if user `daniel`, but still correct)

Hint: The *pwd* command shows the current directory.

cd

cd changes the current directory.

Very useful shortcuts

- `cd -`
  (goes back to the preceding “current” directory)
- `cd`
  (goes to your home directory)

Examples

- `cd ~`
  (same as `cd`)
- `cd ~peter/methodsCL/exercise1/solutions`
- `cd slides`
- `cd ..`

ls

ls displays the contents of a directory.

Examples

- `ls`
  (view contents of current directory)
- `ls ..`
  (view contents of directory above current directory)
- `ls -l`
  (lists more information about each file)
- `ls *.txt`
  (lists only files that end in .txt)
- `ls ~peter/methodsCL/exercise1/solutions/*.txt`

mkdir/rmdir

mkdir creates new directories.
rmdir removes (empty) directories.

Examples

- `cd ~`
- `mkdir newdir`
- `mkdir newdir/subdir`
- `ls newdir`
- `rmdir newdir`
  (wouldn't work because it's not empty)
- `rmdir newdir/subdir`
- `rmdir newdir`

Overview

1 The shell

2 Navigating and manipulating the directory hierarchy

3 The command line

4 Copying, moving and deleting files

5 Permissions

6 Viewing files

7 Standard in and standard out

8 How to get information about commands

9 head/tail, tr, sort, uniq, wc, tee

10 Network

Command-line completion

- hit tab → shell tries to complete a command (or file name)
- if the result is unambiguous, the command (or file name) is completed automatically
- after the first word, it assumes that the next words should refer to files, so it looks for files that start with what has already been typed in
- if several alternatives exist, they are displayed

Examples

- `ged`tab → `gedit`
- `rmdir t`tab → `rmdir test`

Command-line history/editing

- browse through the command line history (preceding commands) using the ↑ and ↓ keys
- commands can be repeated (just hit ⏎) or modified
- Ctrl + a (think “Anfang”) or Home to go to the beginning of the command line
- Ctrl + d deletes the character where the cursor is
- Ctrl + e or End to go to the end of the command line
- Ctrl + k deletes/“kills” anything from cursor to end of line
- you can even paste to the command line, insertion is always at the point where the command line cursor is (irrespective of the mouse pointer!!!)

Starting applications from the command line

- all applications can be started from the command line (instead of clicking an icon)
- e.g. start Adobe's Acrobat Reader

  `/usr/bin/acroread`
- open `file.pdf` using the Acrobat Reader

  `/usr/bin/acroread file.pdf`
- or a text editor `/usr/bin/gedit`
- terminate running processes by typing Ctrl + c

Starting applications from the command line

- you don't need to specify the directory path if your application is located in a directory that is "known" to your computer as holding relevant applications
- `/usr/bin` is usually searched automatically, so it suffices to type `acroread` or `gedit`
- if you don't want the application to "block" your shell, start it by typing `acroread&` or `gedit&` – this sends the new process "to the background" and leaves the shell ready to interact with you

If you kill the shell, background processes will be killed!

Advice

Very useful advice :-)

Save the exact sequence of commands you used in a file. That way, you can repeat the complete process in a second by marking all commands with the mouse, then switching to the shell and pasting all commands to the shell in one go (using the middle mouse button, or Ctrl + Shift + v (Linux) / Ctrl + v (Mac)). This saves an awful lot of time in case you have to repeat the process.

Overview

1 The shell

2 Navigating and manipulating the directory hierarchy

3 The command line

4 Copying, moving and deleting files

5 Permissions

6 Viewing files

7 Standard in and standard out

8 How to get information about commands

9 head/tail, tr, sort, uniq, wc, tee

10 Network

rm

rm removes files.

For some users, the **rm** command has been replaced by `rm -i` by the sysadmins. This causes **rm** to ask back if it really should delete the file

Examples

- `rm *.txt`
- `rm -i file.txt`
  `rm: remove regular file ‘file.txt’?` y/n
- `rm -f file.txt`
  force deletion, don’t ask
- `rm -rf directory`
  recursively remove directory and all its contents without asking

mv/cp

mv moves files, **cp** copies files.
mv can be used to rename files.

For some users, the **mv** and **cp** commands have been replaced by `mv -i` and `cp -i`. This causes them to ask back if they really should overwrite existing files

Examples

- `mv old.txt new.txt`
- `cp old.txt ..`
  copy `old.txt` one directory up
- `mv old.txt subdir`
  - if `subdir` is a directory, `old.txt` is moved to `subdir` and keeps its name
  - else, the file is now called `subdir`

if `subdir` was an existing file, it is now overwritten by `old.txt`!

Overview

1 The shell

2 Navigating and manipulating the directory hierarchy

3 The command line

4 Copying, moving and deleting files

5 Permissions

6 Viewing files

7 Standard in and standard out

8 How to get information about commands

9 head/tail, tr, sort, uniq, wc, tee

10 Network

Permissions

- `ls -l` lists more information about files
- example: `ls -l` in `/corpora/cmd/`

```
-rwxrwxr-x. 1 schmid corpadm  449 18.  Sep 2012 1wpltolex
-rwxr-xr-x. 1 nobody corpadm   62 20.  Apr 1993 5kwic
-rwxr-xr-x. 1 schmid corpadm 1258 18.  Sep 2012 add-missing-blanks.perl
-rwxrwxr-x. 1 schmid corpadm   83 18.  Sep 2012 add-newline.perl
Permissions    Owner   Group  Size   Date/Time   Filename
```

- files and directories can be read, written, and/or executed
- owner: person who decides who can do what with his files
- owner can specify different permissions for three groups of people
  - for himself
  - for a group of users that the owner belongs to
  - for all other users

Permissions

```
Permissions    Owner   Group  Size   Date/Time   Filename

-rwxrwxr-x. 1 schmid corpadm  449 18.  Sep 2012 1wpltolex
-rwxr-xr-x. 1 nobody corpadm   62 20.  Apr 1993 5kwic
-rwxr-xr-x. 1 schmid corpadm 1258 18.  Sep 2012 add-missing-blanks.perl
-rwxrwxr-x. 1 schmid corpadm   83 18.  Sep 2012 add-newline.perl
```

- Here: all entries are files (first character is not “d” or “l”)
- next nine characters: permissions for owner (three char.), group (three char.), and others (three char.)
- here:
  - user `schmid` is allowed to read (“r”), write (“w”), and execute (“x”) the files
  - group members are allowed to read and execute, the first and the fourth file can also be written to
  - others are allowed to read and execute
- for files, *execute* means: execute the code inside them
- for directories, *execute* means: enter the directory

chmod

chmod changes file (or directory) permissions

Examples

- `chmod u+x program.sh`
  make the file `program.sh` executable for its owner (“u”, user)
- `chmod g+r file.txt`
  make the file `file.txt` readable for all users who belong to the group (which group: see output of `ls -l`)
- `chmod o-r file.txt`
  make the file `file.txt` unreadable for all other users (“o”, others)
- `chmod go-rwx`
  compact version: neither group nor others allowed to read, write, or execute

chmod

Advice: give reading permissions as often as possible, it is often very, very helpful to be able to see each other's files!

- . . . when working in groups
- . . . when you want to share resources
- . . . when things don't work and you need help!

groups, chgrp

groups displays the groups a user belongs to
chgrp specifies which group of users has “special” permissions

Examples

- `groups`
  `phonetik sfb732 imsstaff sem0405 synthese`
- `chgrp phonetik NilsHolgerson`
  The group `phonetik` now has extra permissions for file/directory NilsHolgerson

Overview

1 The shell

2 Navigating and manipulating the directory hierarchy

3 The command line

4 Copying, moving and deleting files

5 Permissions

6 Viewing files

7 Standard in and standard out

8 How to get information about commands

9 head/tail, tr, sort, uniq, wc, tee

10 Network

cat

cat lists the contents of a file. It is useful for inspecting short files that are not binarily coded, for instance `.txt` files

Examples

- `cat NilsHolgerson.intro.txt`
  display file NilsHolgerson.intro.txt
- `cat -n NilsHolgerson.intro.txt`
  same, but with line numbers

less

less is similar to **cat**, but it lists the contents page-by-page.
While **less** is running, the following commands are useful:

[ ]	page forward
[b]	page backward
[↵] or [↓]	line forward
[↑]	line backward
[/] `hallo` [↵]	(forward) search for expression “hallo”
[n]	search for next occurrence
[N]	search for preceding occurrence
[q]	end **less** program

Example

- `less NilsHolgerson.intro.txt`

Overview

1 The shell

2 Navigating and manipulating the directory hierarchy

3 The command line

4 Copying, moving and deleting files

5 Permissions

6 Viewing files

7 Standard in and standard out

8 How to get information about commands

9 head/tail, tr, sort, uniq, wc, tee

10 Network

Standard in and standard out

- many commands (e.g. **cat**, **ls**, ...) write to “standard out”
  i.e., they write their results to the shell
- the example commands seen so far got their input from a file
- but they can also read from “standard in”
  i.e., they can take what appears in the shell as an input
- the | operator directs standard out to standard in, i.e. it can combine commands:
- one command writes to standard out; | “pipes” it to standard in, so the next command uses it as input

Combining cat and less

Examples

- `cat stuttgart.txt | less`
- this example is a bit lame because we get the same saying:

  `less stuttgart.txt`
- but using the `-n` option for cat, it makes more sense:

  `cat -n stuttgart.txt | less`

  i.e., we get the same as when using less directly on the file, but with additional line numbers

Redirecting standard out to a file

Anything that goes to standard out can be redirected and written to a file.

Examples

- `cat -n file.txt > file.line.numbers.txt`
  prefix lines with line numbers, write to new file
- `ls *.txt >> filelist`
  append output of ls to a (possibly existing) file called “filelist”

Redirecting standard out to a file

Caution: redirecting to a file which is needed as input destroys the file (produces an empty file; if you're lucky your shell saves you).

Example (`%` is the shell prompt!)

```
% echo hallo > file
% cat file
hallo
% cat file > file
cat: file: input file is output file
```

noclobber

noclobber is a shell variable which can be set to prevent users from overwriting existing files. At IMS, it is by default set for new users (check your `~/.bashrc` file).

- `% echo hello > file`
  create file called “file” with content “hello”
- `% echo hello_again > file`
  `bash: file: cannot overwrite existing file`
  shell refuses to overwrite `file` (only if *noclobber* is set)
- ways out:
  - `% set +C`
    unset noclobber variable only for current shell
  - `% echo hallo >! file`
    using ! noclobber is ignored
  - delete the noclobber command from your `.bashrc` file
    (then noclobber will not be set any more in future shells)

Overview

1 The shell

2 Navigating and manipulating the directory hierarchy

3 The command line

4 Copying, moving and deleting files

5 Permissions

6 Viewing files

7 Standard in and standard out

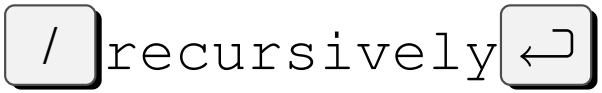
8 How to get information about commands

9 head/tail, tr, sort, uniq, wc, tee

10 Network

Man pages

How do we know about the options like `-n` in `cat -n` or `-rf` in `rm -rf`?

- The usage of most shell commands is documented in the *man(ual) pages*.
- Examples: `man ls`, `man cp`, `man rm` etc.
- Search for particular information in the man page, e. g.: 

- Press 

to quit.
- See man page for **less** to learn how to handle the man pages viewer.

Examples (manpages)

- Which effect has the option `-n` for the commands **cp** and **mv**?
- How can you number only non-empty lines when showing the content of a file with **cat**?
- Which effect have the options `-ltrh` (= `-l -t -r -h`) for the command **ls**?
- How can you search backwards in **less**?

Getting more information about commands

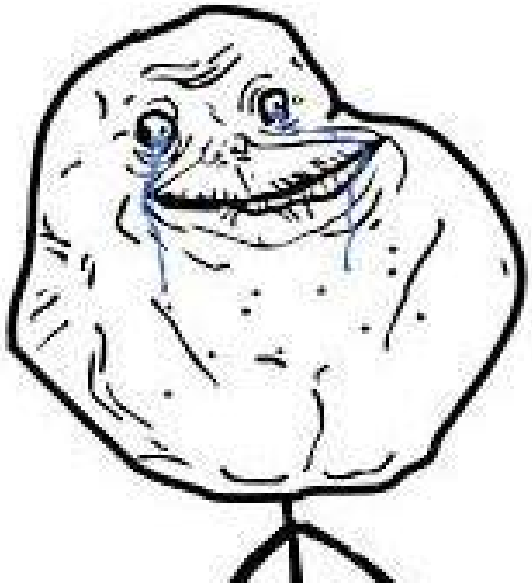
(Remark: Replace `command` in the following examples by the particular command you are looking for.)

- Search for command descriptions: `whatis command`
- Shows the directory of a command: `which command`
- Search for binaries, sources and man pages of a command: `whereis command`
- Search for an expression in man pages and short descriptions of commands: `apropos expression`

whatis

```
root@R2D2:/#
root@R2D2:/# whatis love
love: nichts passendes.
root@R2D2:/#
root@R2D2:/#
root@R2D2:/#
```

Linux forever alone



Overview

1 The shell

2 Navigating and manipulating the directory hierarchy

3 The command line

4 Copying, moving and deleting files

5 Permissions

6 Viewing files

7 Standard in and standard out

8 How to get information about commands

9 head/tail, tr, sort, uniq, wc, tee

10 Network

head & tail

Show the first (**head**) or the last (**tail**) part of files.

- **head**
  - Example: Show the first 20 lines of the file `file.txt`: `head -n 20 file.txt` or `head -20 file.txt`
- **tail**
  - Example: Show the last 20 lines of the file `file.txt`: `tail -n 20 file.txt` or `tail -20 file.txt`

tr

tr translates (i.e. substitutes) symbols.

Examples:

- `cat file.txt | tr ' ' 'X'`
  sometimes useful to make blanks visible
- `tr 'abc' 'def'`
  translate a→d, b→e, c→f
- `tr ' ' '\012'`
  or
  `tr ' ' '\n'`
  translate blanks to newlines (very simple form of tokenizing!!)
- `tr '\011' '\012'`
  translate tabs to newlines
- `tr -d 'x'`
  delete x symbols

sort

sort sorts lines.

Examples:

- `cat *.txt | tr ' ' '\012' | sort`
  sorts words alphabetically after tokenizing
- `sort -nr file.lab`
  numerically sorted (n), order reversed (r) – sometimes useful for phonetic labelfiles to sort by time stamp

uniq

uniq eliminates duplicate lines. Most useful in combination with **sort**.

Examples:

- `cat *.txt | tr ' ' '\012' | sort | uniq`
  gives list of all words in the input, each word once
- `cat *.txt | tr ' ' '\012' | sort | uniq -c`
  same thing, but indicates how many duplicate words there were

wc

wc counts bytes, words, and lines in input.

Examples:

- `wc file.txt`
- `wc -l file.txt`
  output only number of lines
- `cat *.txt | tr ' ' '\012' | sort | uniq | wc -l`
  number of distinct words

tee

tee writes to standard output and at the same time to a file. This is useful to inspect intermediate results when working with pipelines.

Example:

```
cat *.txt |
tr ' ' '\012' | tee 1_after_tr.txt |
sort | tee 2_after_sort.txt |
uniq | tee 3_after_uniq.txt |
wc -l
```

Overview

1 The shell

2 Navigating and manipulating the directory hierarchy

3 The command line

4 Copying, moving and deleting files

5 Permissions

6 Viewing files

7 Standard in and standard out

8 How to get information about commands

9 head/tail, tr, sort, uniq, wc, tee

10 Network

ssh

Remote login on other computers (e. g. login on an IMS computer from home).

Examples:

- `ssh username@somecomputer.ims.uni-stuttgart.de`
- Add the `-X` and `-Y` options to be able to use programs with graphical user interfaces:
  `ssh -X -Y username@somecomputer.ims.uni-stuttgart.de`
  ⚠ This can be really slow!

scp

Copies files from and to other computers (e. g. from IMS computers).

Examples:

- Copy a file from remote computer to your computer:

```
scp␣username@somecomputer.ims.uni-stuttgart.de:
/path/to/file/text.txt␣/home/username/text.txt
```

- Copy a file from your computer to a remote computer:

```
scp␣/home/username/text.txt
username@somecomputer.ims.uni-stuttgart.de:
/path/to/file/text.txt
```

Server

Which server to use for remote login from home?

- Jot down the name of the computer you are working on (see side of the computer case)
- `phoenix.ims.uni-stuttgart.de`
- Your lecturers or supervisors will give you the names of other computers (if you need more power for research projects ;-)).

Server

Remote login from Windows and Mac OS systems

- **Windoof$**
  - Use PuTTY to login to IMS computers from a terminal.
  - Use WinSCP if you want to move files in a graphical program.
- **Mac OS**
  - `ssh` and `scp` can be executed directly from the terminal.
  - Use Macfusion if you want to access IMS computers in the Finder.

?

Questions?

The End

See you in the hands-on session!