Introduction to Linux

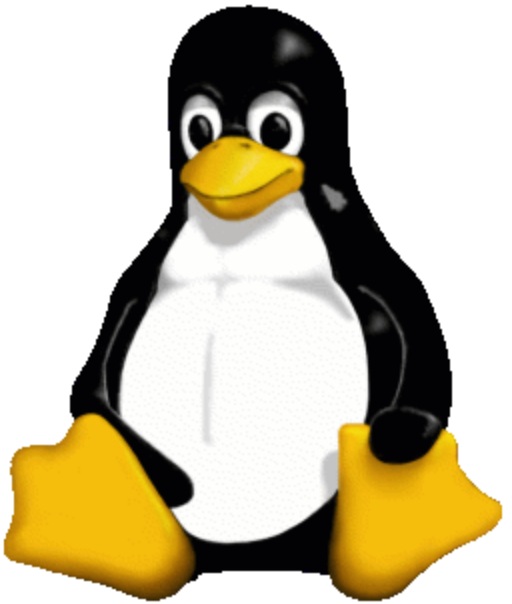
Part I

https://goo.gl/8t6byZ

Agenda

1. What is Linux?
2. Linux interface: GUI vs CLI
3. Connecting to a remote Linux system
4. Linux directory structure
5. Moving and looking around
6. Reading and writing files
7. Organizing files and folders
8. Moving data from/to a remote Linux system

What is Linux?



What is Linux?

What is Linux?

The most common answer you'll hear is:

"Linux is an operating system"

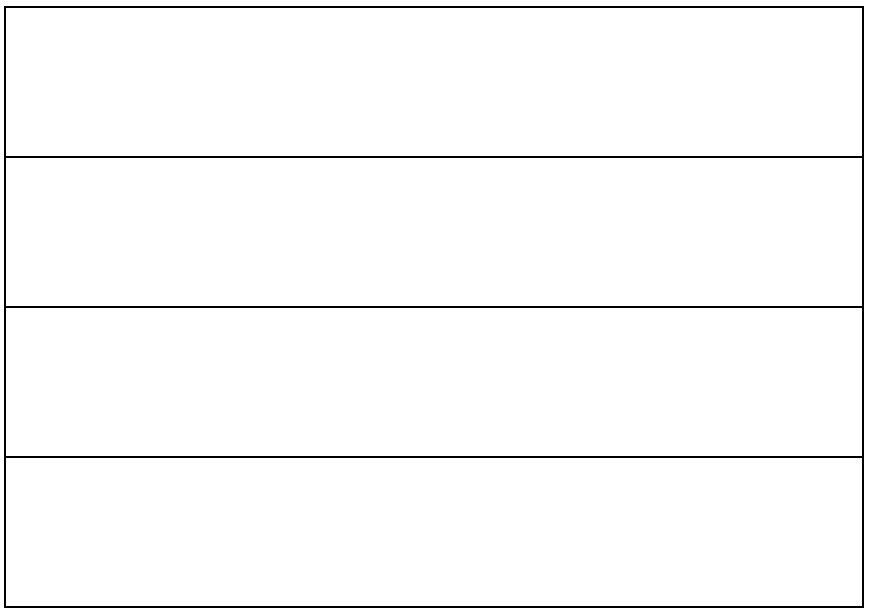
What is Linux?

The most common answer you'll hear is:

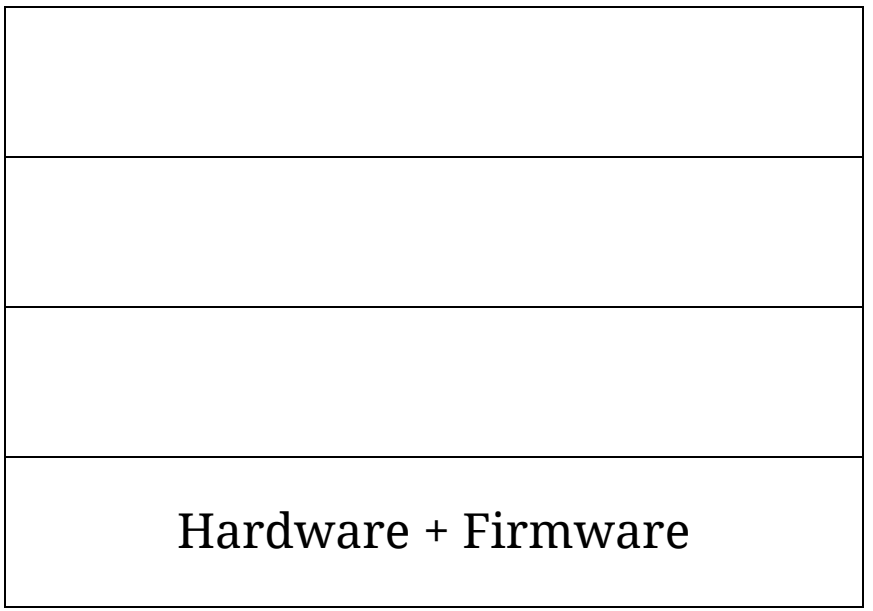
"Linux is an operating system"

But what does this mean?

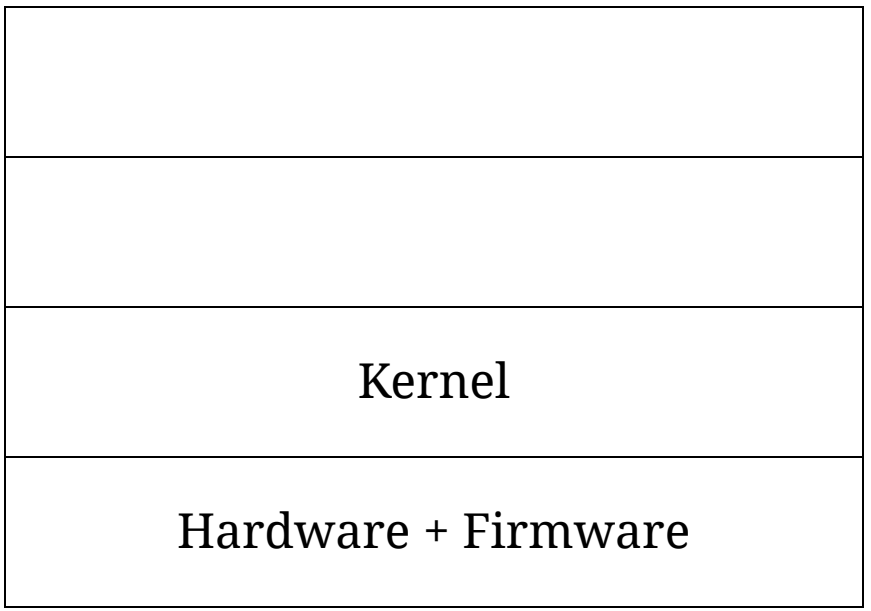
Operating systems



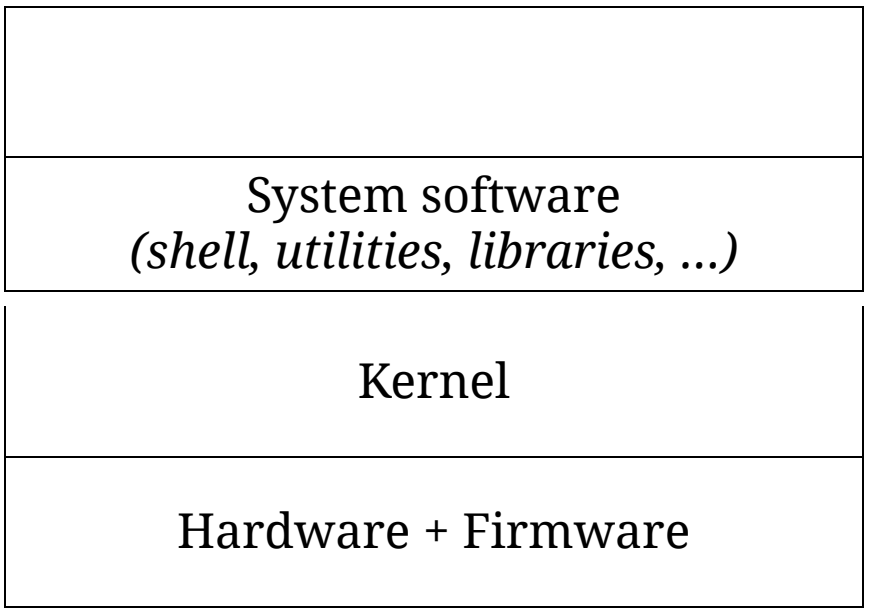
Operating systems



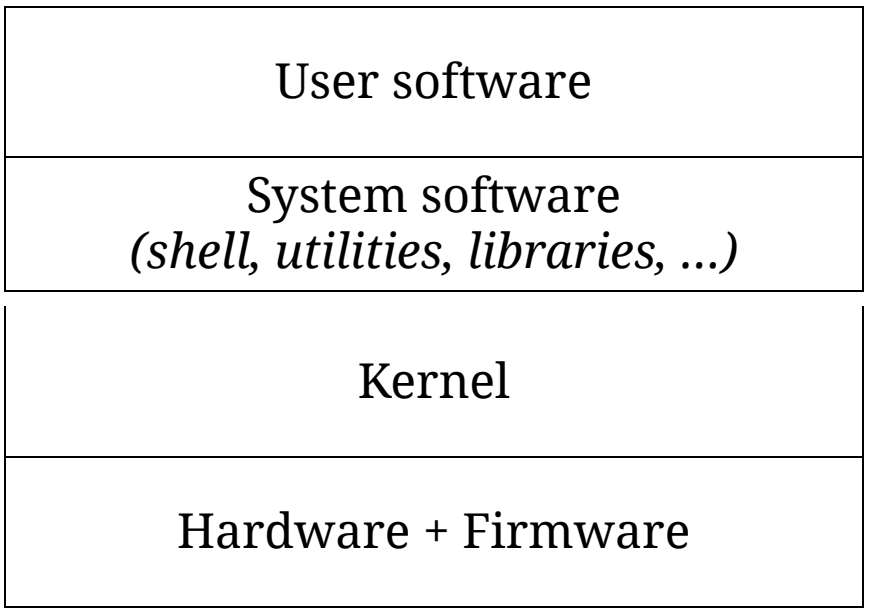
Operating systems



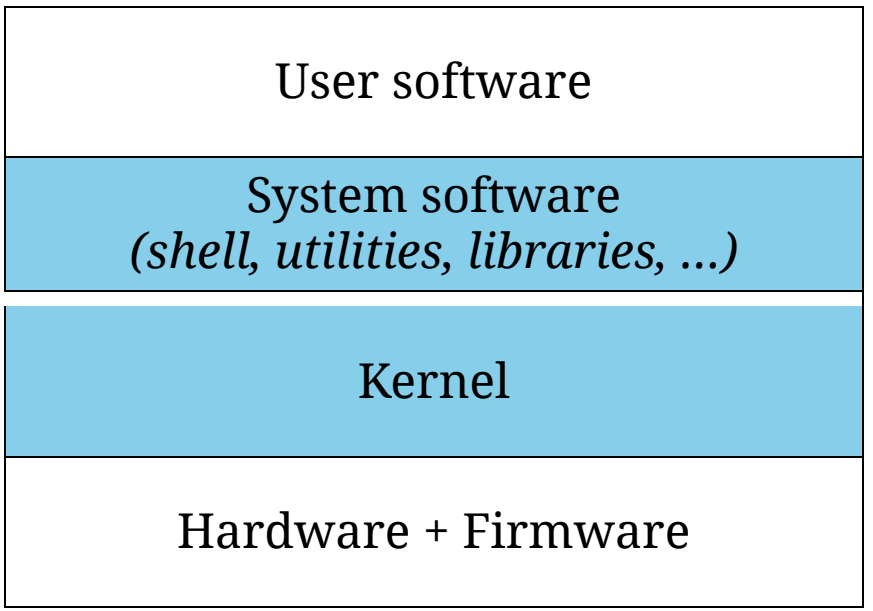
Operating systems



Operating systems

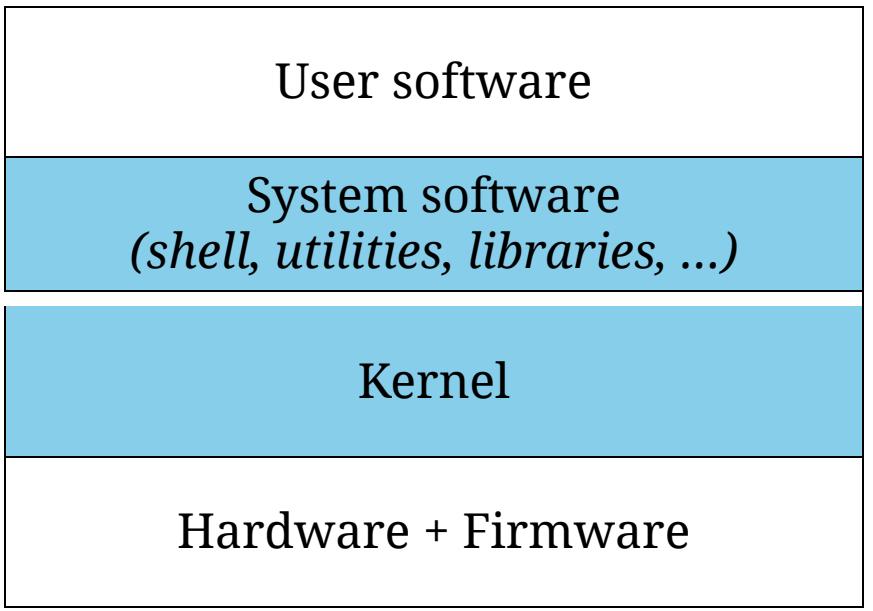


Operating systems



Operating system
Windows, Linux,
MacOS, Android, ...

Operating systems



Linux

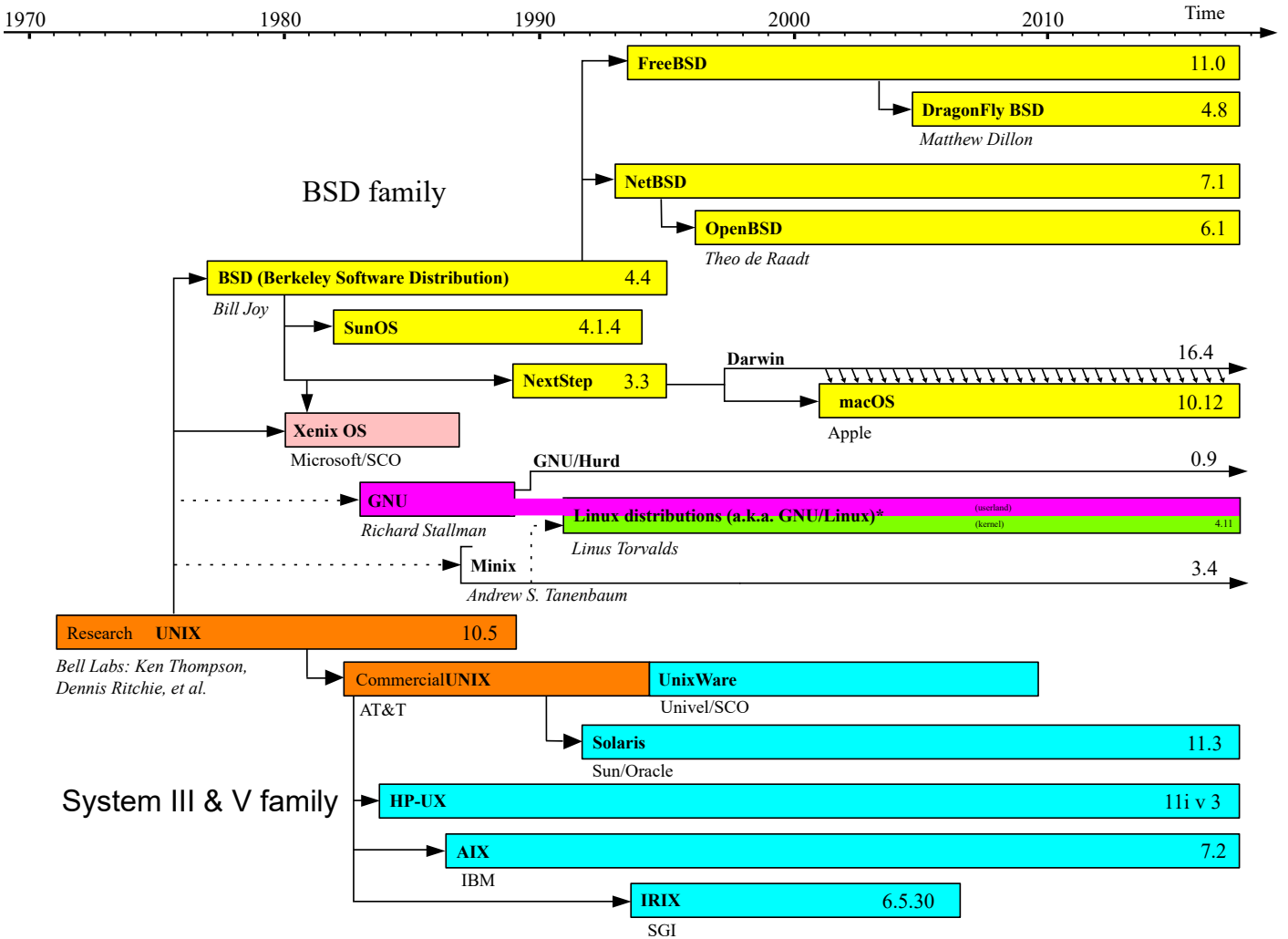
In practice, we call this part "Linux"

Linux? Wait, I also heard "UNIX"?

UNIX is the name of an operating system from 1970 that pioneered concepts that will form the basis of Linux (and other OSes) today.

More importantly, it introduced a set of conventions that its descendents follow. A system that follows them is called "UNIX-like".

Most of what you learn here will easily transfer to other UNIX-like OSes (e.g. macOS).

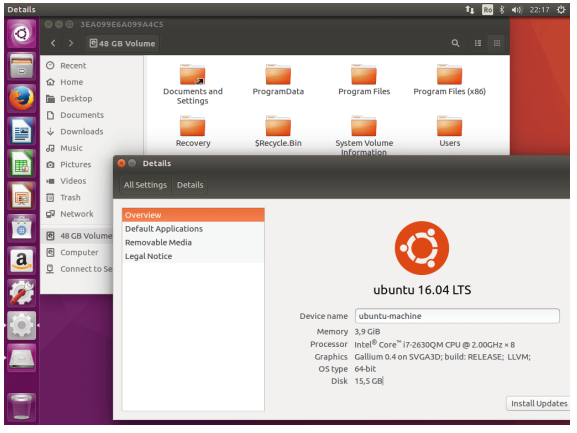


*The penetration of GNU utilities varies between distributions, some projects use GNU's implementation of the Linux kernel (Linux-libre). Some operating systems mentioned here include GNU utilities to a lesser degree.

User Interface

GUI

Graphical Interface



CLI

Command Line

```
Hit:17 http://dl.google.com/linux/chrome/deb stable Release
Hit:18 https://download.docker.com/linux/ubuntu xenial InRelease
Hit:19 http://repository.spotify.com stable InRelease
Hit:20 https://download.sublimetext.com apt/stable/ InRelease
Hit:21 https://deb.nodesource.com/node_7.x xenial InRelease
Hit:24 https://packagecloud.io/slacktechnologies/slack/debian jessie InRelease
Fetched 109 kB in 2s (46.7 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
42 packages can be upgraded. Run 'apt list --upgradable' to see them.
akashev@math67:~
$ ls -la /etc/cron.weekly/
total 40
drwxr-xr-x   2 root root  4096 Jul  3 13:50 .
drwxr-xr-x 165 root root 12288 Jul 26 14:37 ..
-rwxr-xr-x   1 root root   312 Dec 29  2014 0anacron
-rwxr-xr-x   1 root root   730 Apr 13  2016 apt-xapian-index
-rwxr-xr-x   1 root root    86 Apr 13  2016 fstrim
-rwxr-xr-x   1 root root   771 Nov  6  2015 man-db
-rw-r--r--   1 root root   102 Apr  5  2016 .placeholder
-rwxr-xr-x   1 root root   211 Apr 12  2016 update-notifier-common
akashev@math67:~
$
```

Some synonyms:
"Shell", "Terminal", "TTY"

Command Line Interface

```
user@host:~ $ cowsay "Command Line Interface"
 ____________________________
< Command Line Interface >
 ----------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Command Line Interface

```
user@host:~ $ cowsay "Command Line Interface"
 ____________________________
< Command Line Interface >
 ----------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

- Input, output, and commands are **text**.

Command Line Interface

```
user@host:~ $ cowsay "Command Line Interface"
 _____________________________
< Command Line Interface >
 -----------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

- Input, output, and commands are **text**.
- Easy on the computer: can run on any hardware.

Command Line Interface

```
user@host:~ $ cowsay "Command Line Interface"
 _____________________________
< Command Line Interface >
 -----------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

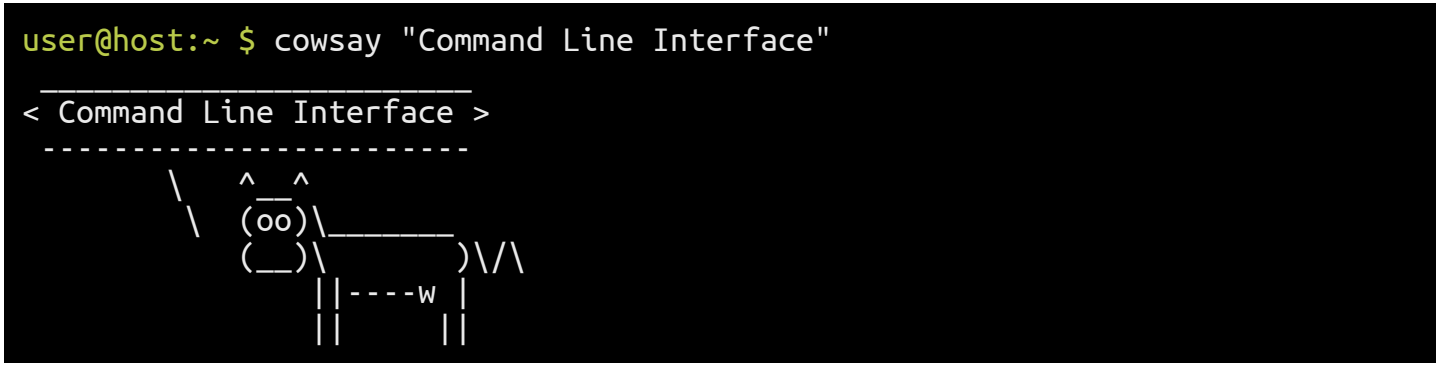
- Input, output, and commands are **text**.
- Easy on the computer: can run on any hardware.
- Network-friendly: a few bytes of text vs realtime stream of images / GUI updates => tool of choice for remote access.

Command Line Interface

```
user@host:~ $ cowsay "Command Line Interface"
 _________________________
< Command Line Interface >
 -------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

- Input, output, and commands are **text**.
- Easy on the computer: can run on any hardware.
- Network-friendly: a few bytes of text vs realtime stream of images / GUI updates => tool of choice for remote access.
- Scripting/automation-friendly: text is easier to manipulate.

Command Line Interface



- Input, output, and commands are **text**.
- Easy on the computer: can run on any hardware.
- Network-friendly: a few bytes of text vs realtime stream of images / GUI updates => tool of choice for remote access.
- Scripting/automation-friendly: text is easier to manipulate.
- Expert-friendly, but *beginner-unfriendly*.

Connecting to a remote Linux system

The standard tool to connect to a remote system is `ssh`.

Acronym:

SSH: **S**ecure **Sh**ell

It securely connects you to a remote system.
Communication is encrypted, both parties are authenticated.

First, you will need to log in to the system.

If your credentials are accepted, it creates a new shell for you.

It is then displayed on your screen and controlled by your keyboard, relayed over the network.

Connecting from MacOS / Linux:

Good news: you already have a terminal and `ssh` of your own!

Connecting from MacOS / Linux:

Good news: you already have a terminal and `ssh` of your own!

First, open the terminal:

- For MacOS, it's accessible from Launchpad, Utilities.
- For Linux GUI, usually look for a program called Terminal.

Connecting from MacOS / Linux:

Good news: you already have a terminal and `ssh` of your own!

First, open the terminal:

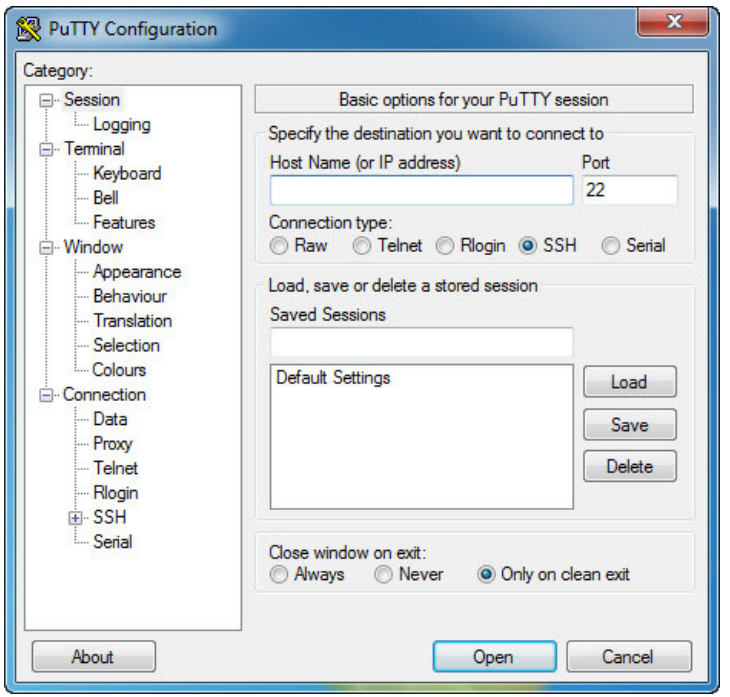
- For MacOS, it's accessible from Launchpad, Utilities.
- For Linux GUI, usually look for a program called Terminal.

Then, you need to input the command to connect to a remote host:

```
local.user@local:~ $ ssh user@remote

[..some mutual* authentication later..]

user@remote:~ $
```

Connecting from Windows:

You will need an SSH client. Standard one: PuTTY



Download the appropriate installer: https://goo.gl/pHFReU

Connecting from Windows:

- Make sure "Connection type: SSH" is selected.
- Put the remote's host name / IP in the form.
- Select "Open"

A terminal window will open..

```
[..some mutual* authentication later..]

user@remote:~ $
```

Mutual authentication?

(and what's up with this side picture?)

Mutual authentication?

SSH authenticates both parties:

- Client to server
  - Username + password
  - Username + cryptographic key
  - Something else!
- Server to client
  - The server has a cryptographic key to prove its identity

Mutual authentication?

SSH authenticates both parties:

- Client to server
  - Username + password
  - Username + cryptographic key
  - Something else!
- Server to client
  - The server has a cryptographic key to prove its identity

The first time you connect, you need to explicitly say you trust the (previously unknown) server.

On subsequent connections, SSH will verify that you are still connecting to a server with the same key, and will warn you before login credentials are transmitted if you aren't.

Mutual authentication?

SSH authenticates both parties:

- Client to server
  - Username + password
  - Username + cryptographic key
  - Something else!
- Server to client
  - The server has a cryptographic key to prove its identity

The first time you connect, you need to explicitly say you trust the (previously unknown) server.

On subsequent connections, SSH will verify that you are still connecting to a server with the same key, and will warn you before login credentials are transmitted if you aren't.

This is called TOFU (**T**rust **O**n **F**irst **U**se).

Mutual authentication

So, the first time you connect to a new server, you should *expect* a warning you need to confirm:

In Linux/MacOS:

```
local.user@local:~ $ ssh user@remote
The authenticity of host 'remote (11.22.33.44)' can't be established.
ECDSA key fingerprint is SHA256:eQZbiUM4qV6ptjc0fN6/pFglj45qaNlXbLCULCTzSGM.
Are you sure you want to continue connecting (yes/no)?
```

Mutual authentication

So, the first time you connect to a new server, you should *expect* a warning you need to confirm:

In Linux/MacOS:

```
local.user@local:~ $ ssh user@remote
The authenticity of host 'remote (11.22.33.44)' can't be established.
ECDSA key fingerprint is SHA256:eQZbiUM4qV6ptjc0fN6/pFglj45qaNlXbLCULCTzSGM.
Are you sure you want to continue connecting (yes/no)? yes
```

Mutual authentication

So, the first time you connect to a new server, you should *expect* a warning you need to confirm:

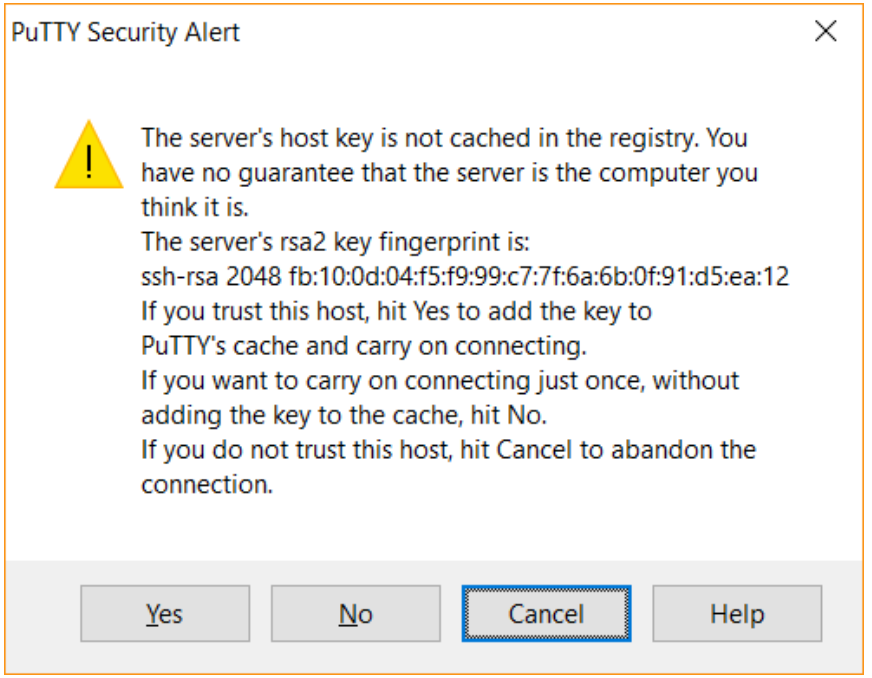
In Linux/MacOS:

```
local.user@local:~ $ ssh user@remote
The authenticity of host 'remote (11.22.33.44)' can't be established.
ECDSA key fingerprint is SHA256:eQZbiUM4qV6ptjc0fN6/pFglj45qaNlXbLCULCTzSGM.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'remote,11.22.33.44' (ECDSA) to the list of known
hosts.

[..some client authentication later..]

user@remote:~ $
```

Mutual authentication

So, the first time you connect to a new server, you should *expect* a warning you need to confirm:

In Windows/PuTTY:



Hands-on time: Connect to a server

Using your Campus account username/password, use SSH/PuTTY to connect to UBELIX at `submit.unibe.ch`.

```
ssh user@submit.unibe.ch
```

A reminder, PuTTY can be obtained from https://goo.gl/pHFReU

Greetings from a shell

Greetings from a shell

After connecting, you will be greeted with something like this:

```
user@remote:~ $
```

Greetings from a shell

After connecting, you will be greeted with something like this:

```
user@remote:~ $
```

What you see is the interface of the **shell**: a text-based interface that allows you to launch other programs with commands.

Greetings from a shell

After connecting, you will be greeted with something like this:

```
user@remote:~ $
```

What you see is the interface of the **shell**: a text-based interface that allows you to launch other programs with commands.

Mnemonic:

It's called a shell **prompt** since it's **prompting** you to enter a command.

The prompt contains a short summary of current state of the shell.

Anatomy of a prompt

The prompt looks like this:

```
user@remote:~ $
```

This may vary slightly from system to system, and is fully configurable, but this is the typical form.

Anatomy of a prompt

The prompt looks like this:

```
user@remote:~ $
```

This may vary slightly from system to system, and is fully configurable, but this is the typical form.

This form of the prompt answers 3 questions:

- Who are you? **Username `user`**
- Where are you? **Hostname `remote`**
- Where in the filesystem are you? **`~`** (explained later)

Anatomy of a prompt

The prompt looks like this:

```
user@remote:~ $
```

This may vary slightly from system to system, and is fully configurable, but this is the typical form.

This form of the prompt answers 3 questions:

- Who are you? **Username `user`**
- Where are you? **Hostname `remote`**
- Where in the filesystem are you? **`~`** (explained later)

Terminating the prompt is (traditionally) a **`$`** character: it delimits where your input goes.

Taking command

The shell expects a textual command; most of the time you type the command and press [ENTER] to commit it.

Let's try this (in slow motion)!

```
user@remote:~ $
```

Taking command

The shell expects a textual command; most of the time you type the command and press [ENTER] to commit it.

Let's try this (in slow motion)!

```
user@remote:~ $ whoami
```

1. Typing in "whoami" as the shell waits

Taking command

The shell expects a textual command; most of the time you type the command and press [ENTER] to commit it.

Let's try this (in slow motion)!

```
user@remote:~ $ whoami
```

1. Typing in "whoami" as the shell waits
2. Pressing [ENTER]. The shell will process the command (launch the program `whoami`)

Taking command

The shell expects a textual command; most of the time you type the command and press [ENTER] to commit it.

Let's try this (in slow motion)!

```
user@remote:~ $ whoami
user
```

1. Typing in "whoami" as the shell waits.
2. Pressing [ENTER]. The shell will process the command — launch the program `whoami`.
3. The program will take over input/output — in this case, it will output your username).

Taking command

The shell expects a textual command; most of the time you type the command and press [ENTER] to commit it.

Let's try this (in slow motion)!

```
user@remote:~ $ whoami
user
user@remote:~ $
```

1. Typing in "whoami" as the shell waits.
2. Pressing [ENTER]. The shell will process the command — launch the program `whoami`.
3. The program will take over input/output — in this case, it will output your username).
4. The program terminates, and control returns to the shell; it shows a new prompt.

Try it!

Here's a few commands for you to try:

```
whoami

echo Hello!

pwd

ls -l

date

sleep 3

clear

history 5
```

Each should do something and return you to the shell prompt. Can you guess what they do?

Note that you can use up/down arrows to access/repeat previous commands.

Safety first, or emergency exits!

So far every command we encountered automatically returned control back to the shell.

But what if a program is stuck, or expecting some input and you're not sure what to do?

Typical shortcuts to stop / quit a program:

- `Ctrl` + `C` (also called **interrupt**)
- `Esc` (from "**escape**")
- `q` (from "**q**uit")
- `Ctrl` + `D` (**end of input**, in case a program is waiting)

If you try those, usually you'll either exit the program or get some hint on how to do it.

`Ctrl` is sometimes denoted as `^`, e.g. `^C` for `Ctrl+C`.

The Working Directory

One of the commands you executed, `pwd`, printed a directory path (of your home directory, by default):

```
akashev@submit01:~ $ pwd
/home/ubelix/math/akashev
```

The Working Directory

One of the commands you executed, `pwd`, printed a directory path (of your home directory, by default):

```
akashev@submit01:~ $ pwd
/home/ubelix/math/akashev
```

Mnemonic:

`pwd` stands for **P**rint **W**orking **D**irectory

The Working Directory

One of the commands you executed, `pwd`, printed a directory path (of your home directory, by default):

```
akashev@submit01:~ $ pwd
/home/ubelix/math/akashev
```

Mnemonic:

`pwd` stands for **P**rint **W**orking **D**irectory

Whenever you use the shell, there is a concept of the current (or "working") directory. This affects how commands search for files and how they interpret paths.

Think of it as of "where" you are: if a server is a building you're in, a working directory is the room you're in within that building.

The Working Directory

One of the commands you executed, `pwd`, printed a directory path (of your home directory, by default):

```
akashev@submit01:~ $ pwd
/home/ubelix/math/akashev
```

Usually, this information is printed in the shell prompt itself, to remind you of the current state.

The Working Directory

One of the commands you executed, `pwd`, printed a directory path (of your home directory, by default):

```
akashev@submit01:~ $ pwd
/home/ubelix/math/akashev
```

Usually, this information is printed in the shell prompt itself, to remind you of the current state.

In this example it's ~, which represents the **home directory**.

The Working Directory

One of the commands you executed, `pwd`, printed a directory path (of your home directory, by default):

```
akashev@submit01:~ $ pwd
/home/ubelix/math/akashev
```

Usually, this information is printed in the shell prompt itself, to remind you of the current state.

In this example it's ~, which represents the **home directory**.

Here's how it would look if you were somewhere else, for example in `/var/log`:

```
akashev@submit01:/var/log $
```

UNIX directory structure

If you're reading this tutorial, you likely already know that files are normally organized into nested "directories" (or "folders"). For example, on Windows you may have such a path:

```
C:\folder\subfolder\file
```

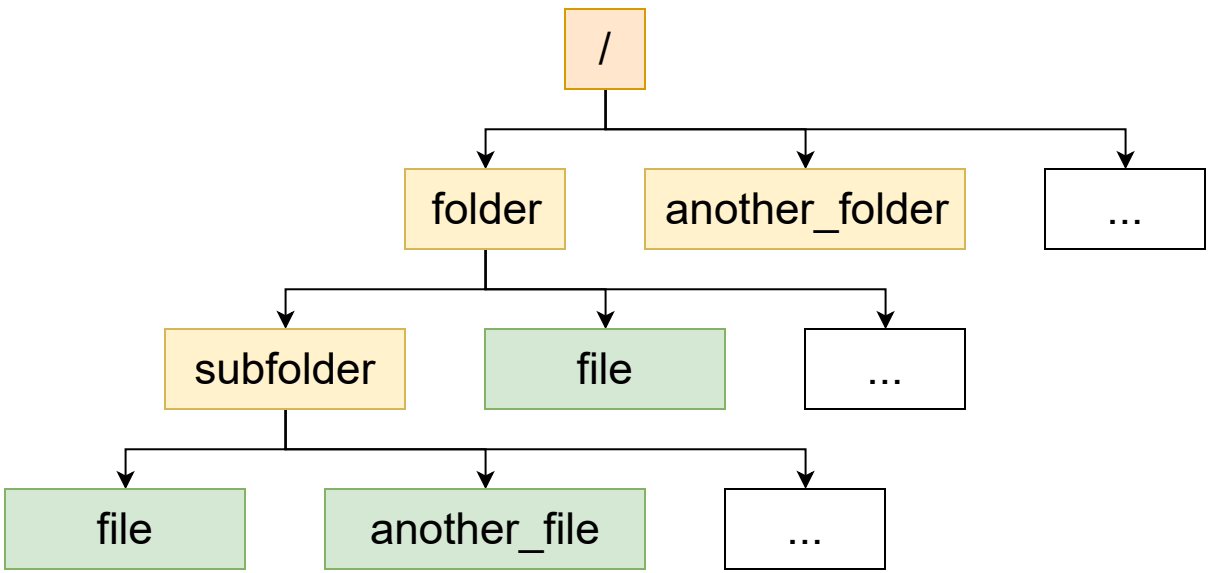
On Linux, paths looks similarly:

```
/home/user/folder/subfolder/file
```

UNIX directory structure

```
/folder/subfolder/file
```

- a file **file**
- inside a directory **subfolder**
- which is inside a directory **folder**
- which itself is inside the **root directory** /



The / as directory separator

Forward slashes (/) separate the folders in the path.
Using multiple is valid, so the following is the same file:

```
/home/user/folder/subfolder/file
///home/user///folder/subfolder//file
```

The / as directory separator

Forward slashes (/) separate the folders in the path.
Using multiple is valid, so the following is the same file:

```
/home/user/folder/subfolder/file
///home/user///folder/subfolder//file
```

A path to a regular file never ends in /, e.g. this is not valid:

```
/home/user/folder/subfolder/file/
```

The / as directory separator

Forward slashes (/) separate the folders in the path.
Using multiple is valid, so the following is the same file:

```
/home/user/folder/subfolder/file
///home/user///folder/subfolder//file
```

A path to a regular file never ends in /, e.g. this is not valid:

```
/home/user/folder/subfolder/file/
```

Directories can be referred to with or without the final /:

```
/home/user/folder/subfolder
/home/user/folder/subfolder/
```

The / as directory separator

Forward slashes (/) separate the folders in the path.
Using multiple is valid, so the following is the same file:

```
/home/user/folder/subfolder/file
///home/user///folder/subfolder//file
```

A path to a regular file never ends in /, e.g. this is not valid:

```
/home/user/folder/subfolder/file/
```

Directories can be referred to with or without the final /:

```
/home/user/folder/subfolder
/home/user/folder/subfolder/
```

Root directory is special: / is its only name.

Absolute and relative paths

If a path starts with `/`, it's an **absolute** path that starts at root:

```
/home/user/folder/subfolder/file
```

Absolute and relative paths

If a path starts with /, it's an **absolute** path that starts at root:

```
/home/user/folder/subfolder/file
```

If it does not, then it's a **relative** path that starts at the current working directory instead of /.

If the current working directory is

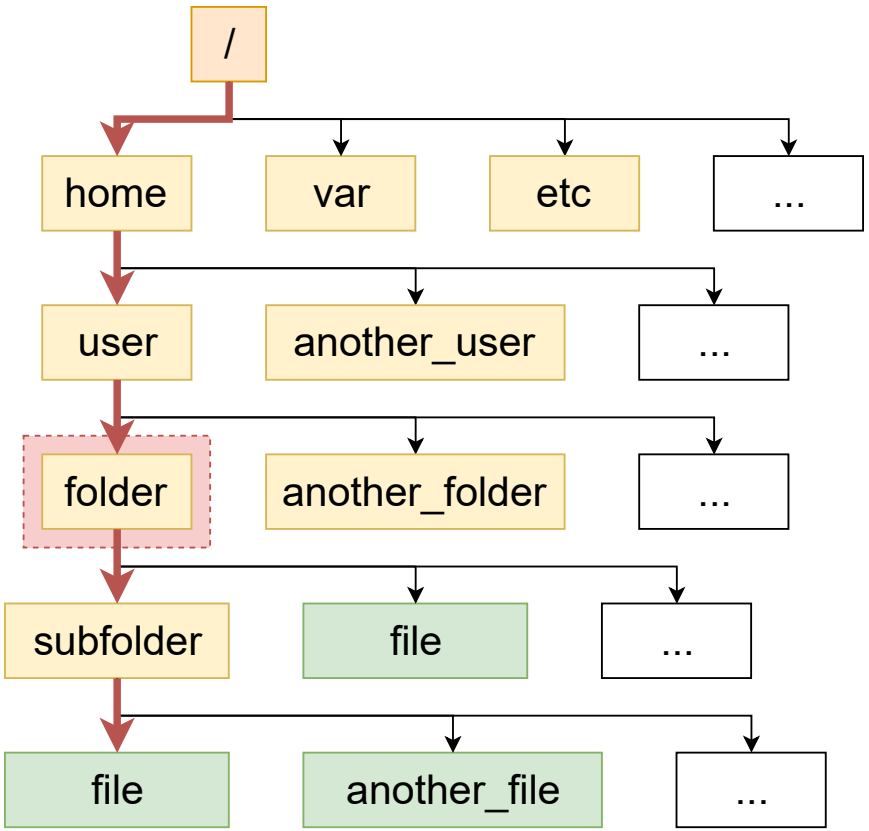
```
/home/user/folder
```

then the following paths point to the same file:

```
/home/user/folder/subfolder/file
subfolder/file
```

Absolute and relative paths

```
/home/user/folder/subfolder/file
subfolder/file
```



Special folders `.` and `..`

There are 2 special folders inside each folder: `.` and `..`

- `.` points to the folder itself.

```
/home/user/folder/subfolder/./file
```

Special folders `.` and `..`

There are 2 special folders inside each folder: `.` and `..`

- `.` points to the folder itself.

```
/home/user/folder/subfolder/./file
```

- `..` points to one folder "up" in the path. At root, it points to root itself.

```
/home/user/another_folder/../folder/file
/home/../../home/user/folder/file
```

Special folders `.` and `..`

There are 2 special folders inside each folder: `.` and `..`

- `.` points to the folder itself.

```
/home/user/folder/subfolder/./file
```

- `..` points to one folder "up" in the path. At root, it points to root itself.

```
/home/user/another_folder/../folder/file
/home/../../home/user/folder/file
```

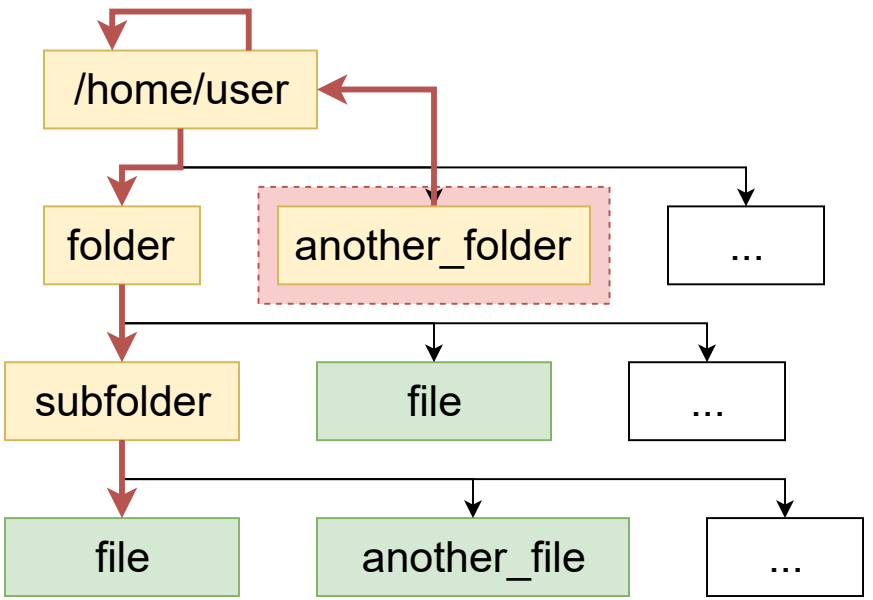
It's mostly important for relative paths:

```
# From /home/user/another_folder
../folder/file
```

Special folders . and . .

From `/home/user/another_folder`

```
/home/user/folder/subfolder/file
.././folder/file
```



Home directories

Each user has a **home directory** assigned.

It acts as your default working directory.

By convention, its path usually starts with `/home/` and ends with your username:

```
/home/<maybe something else>/username
```

It's frequently referred to as ~:

```
/home/username/folder/file
~/folder/file
```

You can even refer to others' home folder with `~username`:

```
/home/someone/file
~someone/file
```

Quiz time! [1/3]

Suppose the following:

```
Username:           userA
Home directory:     /home/userA

Working directory:  /scratch/folder/B
Target:             /scratch/folder/A/a
```

Which of those paths point to the target? (click to reveal)

`~/../scratch/folder/A/a`
`~userA/../../scratch/folder/A/a`
`A/a`
`../A/a`
`/scratch/./folder/A/a`

Quiz time! [2/3]

Suppose the following:

```
Username:           userA
Home directory:     /home/userA

Working directory:  /home/userA/temp
Target:             ../../userB/folder/file
```

Which of those paths point to the target? (click to reveal)

/home/userB/userB/folder/file
/home/userB/folder/file
~/folder/file
~/../userB/folder/file
~userB/folder/file

Quiz time! [3/3]

Suppose the following:

```
Username:           userA
Home directory:     /home/userA

Working directory:  /home/userA/folder
Target:             /home/userA/folder/file
```

Which of those paths point to the target? (click to reveal)

`file`
`./file`
`~/file`
`~/folder/file`
`/home/userA/folder/subfolder/../file`

Preparing for training

Please execute the following command to add the exercises to your home folder:

```
$ wget https://scits.math.unibe.ch/script -O - | /bin/bash
```

This should be the only time you don't understand what you're doing; and by the end of Part II you should understand it.

Moving around

Now that we know:

- Files and directories are organised in a tree
- There's a "current"/working directory that we are in

we need to learn to move around in that tree.

Moving around

Now that we know:

- Files and directories are organised in a tree
- There's a "current"/working directory that we are in

we need to learn to move around in that tree.

For that, we need the `cd` command:

```
user@remote:~ $ cd scits-training
user@remote:~/scits-training $ pwd
/home/username/scits-training
user@remote:~/scits-training $
```

Acronym:

`cd` stands for "**C**hange **D**irectory"

Moving around

The general format of the command is `cd DESTINATION`, where `DESTINATION` is a path (relative or absolute) to a directory.

```
user@remote:~ $ cd scits-training
user@remote:~/scits-training $ cd /usr/local/bin
user@remote:/usr/local/bin $
```

Moving around

The general format of the command is `cd DESTINATION`, where `DESTINATION` is a path (relative or absolute) to a directory.

```
user@remote:~ $ cd scits-training
user@remote:~/scits-training $ cd /usr/local/bin
user@remote:/usr/local/bin $
```

To go "back up", one uses the special `..` directory:

```
user@remote:/var/local/bin $ cd ..
user@remote:/usr/local $ cd ../..
user@remote:/ $
```

Moving around

The general format of the command is `cd DESTINATION`, where `DESTINATION` is a path (relative or absolute) to a directory.

```
user@remote:~ $ cd scits-training
user@remote:~/scits-training $ cd /usr/local/bin
user@remote:/usr/local/bin $
```

To go "back up", one uses the special `..` directory:

```
user@remote:/var/local/bin $ cd ..
user@remote:/usr/local $ cd ../..
user@remote:/ $
```

To go to your home directory, you can use ~:

```
user@remote:/ $ cd ~
user@remote:~ $
```

cd shortcuts

There are two useful tricks when using `cd`:

"`cd -`" goes back to the previous directory you were in:

```
user@remote:~ $ cd -
user@remote:/ $
```

And "`cd`" without arguments goes to your home folder:

```
user@remote:/ $ cd
user@remote:~ $
```

Tab-completion

This is a good point to introduce a helpful CLI tool: **tab completion**

When entering a command, you can press the `[Tab]` key to suggest a command, or path, based on already entered input.

```
user@remote:~ $ cd scits-training/a
```

Pressing `[Tab]` now completes the name, since it's the only one that matches the beginning:

```
user@remote:~ $ cd scits-training/animals/
```

(continues on next slide)

Tab-completion

```
user@remote:~ $ cd scits-training/animals/
```

Pressing `[Tab]` once again won't change anything, since there are mutiple choices for completion; however, if it is pressed again, it shows possibilities:

```
user@remote:~ $ cd scits-training/animals/
Aardvark/ Badger/
user@remote:~ $ cd scits-training/animals/
```

Tab-completion

```
user@remote:~ $ cd scits-training/animals/
```

Pressing `[Tab]` once again won't change anything, since there are mutiple choices for completion; however, if it is pressed again, it shows possibilities:

```
user@remote:~ $ cd scits-training/animals/
Aardvark/ Badger/
user@remote:~ $ cd scits-training/animals/
```

The shell needs to know the next letter to proceed. So, we type only "A" and press `Tab` again:

```
user@remote:~ $ cd scits-training/animals/A
```

Tab-completion

```
user@remote:~ $ cd scits-training/animals/
```

Pressing `[Tab]` once again won't change anything, since there are mutiple choices for completion; however, if it is pressed again, it shows possibilities:

```
user@remote:~ $ cd scits-training/animals/
Aardvark/ Badger/
user@remote:~ $ cd scits-training/animals/
```

The shell needs to know the next letter to proceed. So, we type only "A" and press `Tab` again:

```
user@remote:~ $ cd scits-training/animals/Aardvark/
```

Tab-completion

```
user@remote:~ $ cd scits-training/animals/
```

Pressing `[Tab]` once again won't change anything, since there are mutiple choices for completion; however, if it is pressed again, it shows possibilities:

```
user@remote:~ $ cd scits-training/animals/
Aardvark/ Badger/
user@remote:~ $ cd scits-training/animals/
```

The shell needs to know the next letter to proceed. So, we type only "A" and press `Tab` again:

```
user@remote:~ $ cd scits-training/animals/Aardvark/
user@remote:~/scits-training/animals/Aardvark/ $
```

Looking around

To look around in a UNIX filesystem, you use the `ls` command:

```
user@remote:~/scits-training/animals/Aardvark/ $ ls
description  empty_file  subfolder
```

Mnemonic:

`ls` stands for **list**

This lists the names for contents of the working directory.

Looking around

To look around in a UNIX filesystem, you use the `ls` command:

```
user@remote:~/scits-training/animals/Aardvark/ $ ls
description  empty_file  subfolder
```

Mnemonic:

`ls` stands for **list**

This lists the names for contents of the working directory.

We can specify another folder to look at:

```
user@remote:~/scits-training/animals/Aardvark/ $ ls ../Badger/
Arctonyx  Meles  Mellivora  Melogale  Mydaus
```

Looking around (in depth)

To show more information, we can use the `-l` (for **l**ong) flag:

```
user@remote:~/scits-training/animals/Aardvark/ $ ls -l
total 25640
-rw-r--r-- 1 username groupname 26214400 Aug 28 18:20 big_file
-rw-r--r-- 1 username groupname      754 Aug 25 17:55 description
-rw-r--r-- 1 username groupname        0 Aug 28 16:51 empty_file
drwxr-xr-x 2 username groupname     4096 Aug 28 16:52 subfolder
```

Important information from this output:

- `-rw-r--r--` is called the **mode** (explained in Part II).
  - `d` denotes **directory** in this example.
  - `rw-r--r--` deals with permissions for the files.
- `username` and `groupname` are **owners** of the file.
- The number after `groupname` is the **size** (in bytes) of the file.
  - Important: for folders, it's not the size of all contents.
- The date/time after the size is the **modification date**.

Looking around (as puny humans)

One can use the flag `-h` (for **h**uman-readable) for more familiar size units:

```
user@remote:~/scits-training/animals/Aardvark/ $ ls -l -h
total 26M
-rw-r--r-- 1 username groupname  25M Aug 28 18:20 big_file
-rw-r--r-- 1 username groupname  754 Aug 25 17:55 description
-rw-r--r-- 1 username groupname    0 Aug 28 16:51 empty_file
drwxr-xr-x 2 username groupname 4096 Aug 28 16:52 subfolder
```

Single-letter flags in commands can often be combined:

```
user@remote:~/scits-training/animals/Aardvark/ $ ls -lh
total 26M
-rw-r--r-- 1 username groupname  25M Aug 28 18:20 big_file
-rw-r--r-- 1 username groupname  754 Aug 25 17:55 description
-rw-r--r-- 1 username groupname    0 Aug 28 16:51 empty_file
drwxr-xr-x 2 username groupname 4096 Aug 28 16:52 subfolder
```

Looking around (into hidden corners)

Another often-used flag is `-a` (for **a**ll): it lists contents with names that start with a dot `.` which are normally hidden in UNIX.

```
user@remote:~/scits-training/animals/Aardvark/ $ ls -a
.  ..  big_file  description  empty_file  .hidden  subfolder
```

As usual, it can be combined with others:

```
user@remote:~/scits-training/animals/Aardvark/ $ ls -lah
total 26M
drwxr-xr-x 2 username groupname 4096 Aug 28 16:52 .
drwxr-xr-x 2 username groupname 4096 Aug 28 16:52 ..
-rw-r--r-- 1 username groupname  25M Aug 28 18:20 big_file
-rw-r--r-- 1 username groupname  754 Aug 25 17:55 description
-rw-r--r-- 1 username groupname    0 Aug 28 16:51 empty_file
drwxr-xr-x 2 username groupname 4096 Aug 28 16:52 subfolder
-rw-r--r-- 1 username groupname    0 Aug 28 16:51 .hidden
```

Looking around (in orderly fashion)

By default, files are ordered by name.

This behavior can be changed with flags; here are some examples:

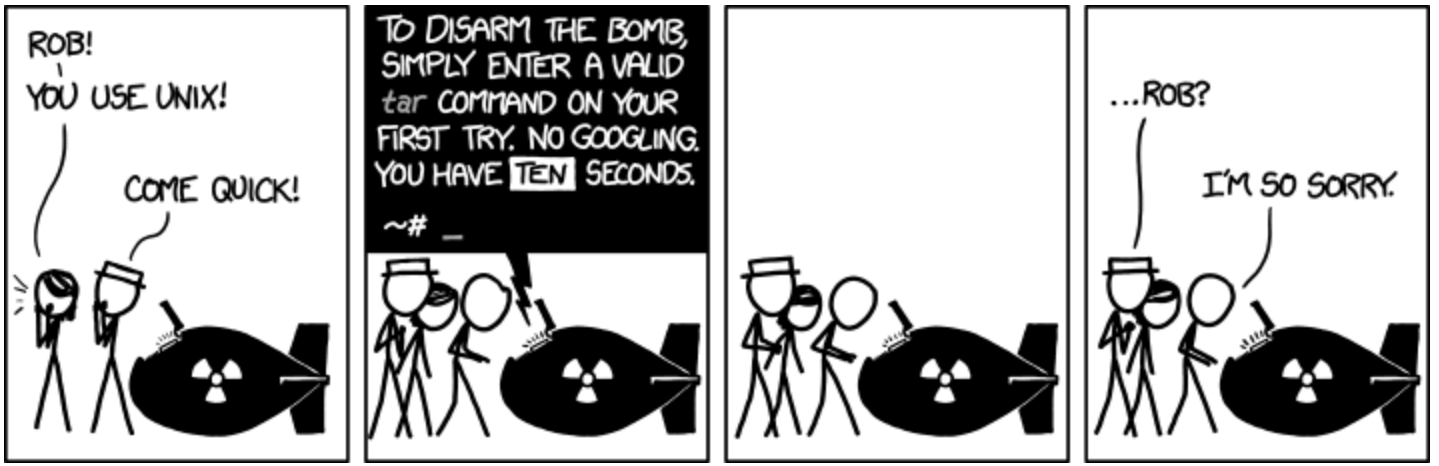
- `-r` **r**everses the sort order.
- `-S` sorts files by **s**ize.
- `-t` sorts files by modification **t**ime.
- `-X` sorts files by filename e**x**tension, e.g. `png` in `image.png`.

As usual, this can be combined with the previous ones.

Exercise:

List files in `Aardvark` by increasing size.

I'm never going to remember this!



Good news: you don't have to.

As long as you remember the command's name, you can look up its correct usage from the terminal itself.

Image credit: https://xkcd.com/1168/

Getting help

Some common methods of getting help:

- Many programs support **`--help` flag** to print out their usage instructions:

```
user@remote:~/scits-training/animals/Aardvark/ $ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
[...]
```

Getting help

Some common methods of getting help:

- For most programs, you can look up their **manual file** with `man`:

```
user@remote:~/scits-training/animals/Aardvark/ $ man ls
```

Instead of just outputting the text and returning, you'll enter a mode for showing long files.

Look around using arrow keys and `PgDn`/`PgUp`.

Remember the hints on how to exit (here, it's `q`).

You can search a `man` page for "something" with `/something` and just `/` to go to the next find.

Getting help

Some common methods of getting help:

- Some commands are not separate programs, but are **built into the shell**, e.g. `cd`. For those, you can use `help`:

```
user@remote:~/scits-training/animals/Aardvark/ $ help cd
```

Getting help

Some common methods of getting help:

- Some commands are not separate programs, but are **built into the shell**, e.g. `cd`. For those, you can use `help`:

```
user@remote:~/scits-training/animals/Aardvark/ $ help cd
```

You can see what `help` can help with as well:

```
user@remote:~/scits-training/animals/Aardvark/ $ help
```

Try out `man`

Try opening the manual for `ls`:

```
user@remote:~/scits-training/animals/Aardvark/ $ man ls
```

Reminders:

- You can search a `man` page for "something" with `/something` and `n` to go to the next find.
- To exit, you can use `q`.

Exercise:

Try searching for the meaning of `-R` flag, and try to use it.

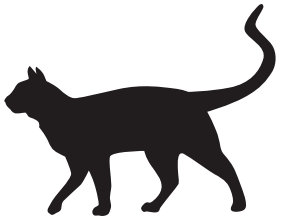
Reading files

We know how to look around the filesystem (with `ls`) and how to move around (with `cd`).

However, we still need to access the contents of files.

There are many ways to do that, I'll show a few more common ones.

Simple file reading



The simplest program to read the file is `cat`

```
user@remote:~/scits-training/animals/Aardvark/ $ ls
big_file  description  empty_file  naming  subfolder
user@remote:~/scits-training/animals/Aardvark/ $ cat description
The aardvark (ARD-vark; Orycteropus afer) is a medium-sized, burrowing,
[...]
```

Mnemonic:

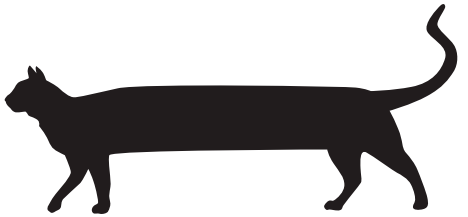
`cat` comes from the word "con**cat**enate",
which means joining things together in a series.

Exercise:

What happens if we call `cat` with two filenames?

```
cat description naming
```

File is too long!



Sometimes a file is too long to be comfortably read with `cat`

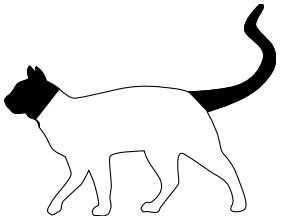
```
user@remote:~/scits-training/animals/Aardvark/ $ cd ../../numbers/
user@remote:~/scits-training/numbers/ $ cat hundred
1
2
[...]
99
100
```

A hundred lines is too much to fit into the terminal window.

While you can scroll to look through the output, sometimes files are much longer than that.

We can display only parts of the file, or use a program that allows to navigate a file.

Parts of a cat?



If a cat is too long, perhaps we only need to look at its beginning (`head`) or end (`tail`):

```
user@remote:~/scits-training/numbers/ $ head hundred
1
[...]
10
```

```
user@remote:~/scits-training/numbers/ $ tail hundred
91
[...]
100
```

Those commands display the first and last 10 lines of a file, respectively.

Mnemonic:

Remembering `cat` together with `head` and `tail` may help.

Self-help test

Of course, you can look up other options with the self-help methods like `man`.

Exercise:

Use one of the help methods (`man head` or `head --help`) to learn how to display 5 lines instead of 10 with `head`.

Hint: it will be a flag that should go before the filename.

Self-help test

Of course, you can look up other options with the self-help methods like `man`.

Exercise:

Use one of the help methods (`man head` or `head --help`) to learn how to display 5 lines instead of 10 with `head`.

Hint: it will be a flag that should go before the filename.

Answer: `-n 5`, `-n5` or `--lines=5`

```
user@remote:~/scits-training/numbers/ $ head -n 5 hundred
1
2
3
4
5
```

The file is too long, show less

One way to navigate a big file is `less`:

```
user@remote:~/scits-training/numbers/ $ less hundred
```

You will recognize this interface, since `man` also uses `less`.

Commands to try:

- **Arrow keys** to scroll line by line
- **PgUp** / **PgDn** to scroll screen by screen
- **/something** to search for "something"
- **n** to go to next found "something", **N** to go back
- **>** to go to the end of the file, **<** to go to the beginning
- **h** to show help
- **q** to quit

Modifying files

Besides reading, we need to be able to create and modify files.

There are many editors available, and which one is "best" can lead to hot debate.

We will mention and briefly explain two editors that are likely to be installed on any system you encounter nowadays.

- `nano`
- `vim`

nano


user@remote:~/scits-training/numbers/ $ nano hundred
GNU nano 2.5.3 File: hundred
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
[Read 100 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line

nano

`nano` is a small and simple editor which helpfully shows its commands at the bottom (reminder, `^` means `Ctrl`):

You can use arrow keys to move around, input text as normal from where the cursor is.

Key commands:

- `Ctrl` + `W` "**w**here is" for searching the file
- `Ctrl` + `O` "write **o**ut" to save changes
- `Ctrl` + `X` "e**x**it" to get back to the shell

Try nano

Exercise:

1. Open a new file, `ten`, with `nano`:

```
user@remote:~/scits-training/numbers/ $ nano ten
```

2. Add numbers from 1 to 10 to it, on separate lines
3. Save and exit `nano`
4. Verify what's in the file using `cat`

vim

`vim` (or, technically, "Vi IMproved") is one of two "Swiss knife" editors that most Linux professionals prefer to use (the other one being `emacs`).

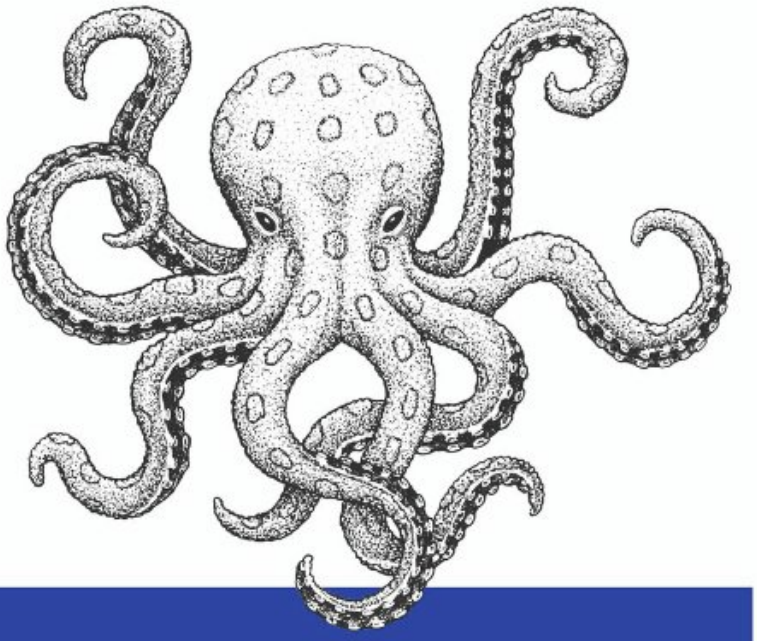
`vim` is available almost everywhere, and with proper configuration can do very sophisticated things.

With power comes complexity, but for basic editing one doesn't have to remember a lot.

If you wish to (later) explore `vim`, you can go through its built-in tutorial:

```
vimtutor
```

Just memorize these fourteen contextually dependant instructions



Exiting Vim

Eventually

O RLY?

@ThePracticalDev

Organizing files and folders

To recap, you should now be able to:

- Navigate the file tree (with `cd`)
- List folder contents (with `ls`)
- Read and write files (with `nano`)

Our goal now is:

- Make new folders
- To move and copy files and folders around
- Delete files and folders

Creating new folders

To create new folders, use the `mkdir` command:

```
user@remote:~/scits-training/numbers/ $ cd ..
user@remote:~/scits-training/ $ ls
animals  numbers
user@remote:~/scits-training/ $ mkdir new-folder
user@remote:~/scits-training/ $ ls
animals  new-folder  numbers
```

Mnemonic:

`mkdir` stands for **make dir**ectory

Exercise:

1. Create `new-folder` as shown above
2. Create directory `subfolder` inside it
3. Verify with `ls`

Creating new folders

`mkdir` will fail if the folder already exists:

```
user@remote:~/scits-training/ $ mkdir new-folder
mkdir: cannot create directory ‘new-folder’: File exists
```

Using it with `-p` means "create if needed", and also works with chains of directories:

```
user@remote:~/scits-training/ $ mkdir -p new-folder/subfolder/subsubfolder
user@remote:~/scits-training/ $ ls -R new-folder
new-folder:
subfolder

new-folder/subfolder:
subsubfolder

new-folder/subfolder/subsubfolder:
user@remote:~/scits-training/ $
```

Moving files

Move operations can be broken down into two cases:

1. Moving files and folders between folders:

   `folder1/something` → `folder2/something`

2. Renaming files and folders:

   `something` → `other`

   Technically, it's "moving" from old name to new.

Both cases are served with the `mv` command.

Mnemonic:

`mv` stands for **move**

Preparing for exercises

```
user@remote:~/scits-training/ $ cd moving
user@remote:~/scits-training/moving $ ls
source  destination
user@remote:~/scits-training/moving $ ls source
A1  A10  A11  A12  A2  A3  A4  A5  A6  A7  A8  A9  subfolder
user@remote:~/scits-training/moving $ ls source/subfolder
B1  B2  B3  B4  B5  B6  B7  B8  B9
user@remote:~/scits-training/moving $ ls destination
user@remote:~/scits-training/moving $
```

Moving files

To move something to another folder: **`mv NAME DESTINATION`**, as long as the **`DESTINATION` is a directory that exists**.

```
user@remote:~/scits-training/moving $ mv source/A2 destination
```

You can specify multiple things to move at the same time, including folders:

```
$ mv source/A3 source/subfolder destination
```

Moves both `source/A3` and `source/subfolder` into `destination`.

Exercise:

Move `subfolder` back into `source`

Renaming

Renaming is easy: `mv OLDNAME NEWNAME`, if `NEWNAME` is *not* a directory.

For example, let's rename `destination` to `dest`:

```
user@remote:~/scits-training/moving $ mv destination dest
```

If you're renaming something in another folder, you must specify the path twice:

```
$ mv source/A4 source/A40
```

Exercise:

1. Rename `dest` back into `destination`
2. Rename `source/subfolder/B1` into `source/subfolder/B10`

Move + rename

Exercise:

Try the following:

```
user@remote:~/scits-training/moving $ mv source/A5 A50
```

Use `ls` to understand what happened (`-R` may help)

Move + rename

Exercise:

Try the following:

```
user@remote:~/scits-training/moving $ mv source/A5 A50
```

Use `ls` to understand what happened (`-R` may help)

Answer: Since there is no path for the second name, it moved into the current directory and got renamed:

`~/scits-training/moving/source/A5`

↓

`~/scits-training/moving/A50`

Copying

Copying is done with `cp`

Mnemonic:

`cp` stands for **copy**.

Syntax is the same:

- For copying to another directory, `cp NAME DESTINATION`
- For copying to another name, `cp OLDNAME NEWNAME`

Exercise:

1. Copy `source/A6` and `source/A7` into `destination`
2. Copy `source/A6` into `source/A66`

Copying folders

`cp`, unlike `mv`, will not copy directories by default:

```
$ cp source/subfolder destination
cp: omitting directory 'source/subfolder'
```

You need to use `-R` to copy folders together with their content

```
$ cp -R source/subfolder destination
```

Mnemonic:

-R stands for **r**ecursive

Deleting

To remove files or folders, use `rm`

Mnemonic:

`rm` stands for **rem**ove

- `rm NAME` to remove a file
- `rm -r FOLDER` to remove a folder

You can pass several names at once:

```
$ rm destination/subfolder/B1 destination/subfolder/B2
```

`rm` is unrecoverable!

When you delete files and folders with `rm`, you should be aware that there is no concept of "Trash".

Anything you delete (or overwrite) is lost with no easy way to recover.

You can use a flag `-i` to ask before any destructive operation.

```
user@remote:~/scits-training/moving $ cp -i -R source/subfolder destination
cp: overwrite 'destination/subfolder/B1'?
```

On the other hand, sometimes you want to override those confirmations, especially for `rm` – you can do it with `-f`.

Mnemonic:

`-i` stands for **interactive**
`-f` stands for **force**

Wildcards

There are many A-files in `source`:

```
user@remote:~/scits-training/moving $ ls source
A1  A10  A11  A12  A40  A6  A66  A7  A8  A9  subfolder
```

We may want to copy them all at once. We can use wildcards:

- **`*`** in a name means "any amount of any characters"
  - For example, **`A*`** can mean `A`, `A1` and `A10`
- **`?`** in a name means "any single character"
  - For example, **`A?`** can mean `A1`, `A6` but not `A10`

The wildcards will **not jump through directories**:

- **`*1`** can mean `A1`, `A11`, but not `subfolder/B1`
- **`*/*`** can match `subfolder/B1`

Wildcard quiz (1/3)

Which of the following names match the pattern **`A*a*`**

`AAa`
`A/a`
`aaA`
`CBAcba`
`abcABC`

Wildcard quiz (2/3)

Which of the following names match the pattern `A?a?`

`AAa`
`AAaa`
`Aaaa`
`AAaaa`
`aAAaa`

Wildcard quiz (3/3)

Which of the following patterns match the name **`A110`**

`A*`
`*`
`*A*`
`*A`
`A???`

Using wildcards

Putting a name with a wildcard is equivalent to putting several names:

```
$ cp source/A6* destination
```

is equivalent to

```
$ cp source/A6 source/A66 destination
```

So, you can use wildcards in any command that expects multiple files.

Try wildcards

Use wildcards to do the following, from `~/scits-training/moving`:

Exercise:

1. List all files starting with `A` inside `source/` (use `ls` with a pattern).
2. Copy all files starting with `B` from `source/subfolder` into `destination`.
3. Move all files starting with `A1` from `source` into `destination`.
4. Delete all files starting with `A` from `destination`.

Moving data in and out

So far we have moved the data around on the system itself.

It doesn't help if you want to load external data or download the results of your programs.

Moving data in and out

So far we have moved the data around on the system itself.

It doesn't help if you want to load external data or download the results of your programs.

Perhaps, it's your own system and you have access to cloud storage or external storage devices.

Sometimes, you have a shared network folder between your computer and the target system.

But how to do it, if your only interface to the server is SSH?

Moving data in and out

So far we have moved the data around on the system itself.

It doesn't help if you want to load external data or download the results of your programs.

Perhaps, it's your own system and you have access to cloud storage or external storage devices.

Sometimes, you have a shared network folder between your computer and the target system.

But how to do it, if your only interface to the server is SSH?

We will cover two ways:

1. Downloading data from the Internet with `wget`
2. Copying data between computers with `scp`

Downloading from the shell

Sometimes, the data you need is a file on the Internet.

`wget` is the simplest-to-use tool for it:

```
user@remote:~/scits-training/moving $ cd ..
user@remote:~/scits-training/ $ wget https://example.com/
--2017-09-12 12:00:00--  https://example.com/
Resolving example.com (example.com)... 93.184.216.34
Connecting to example.com (example.com)|93.184.216.34|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1270 (1.2K) [text/html]
Saving to: ‘index.html’

100%[======================================>] 1,270       --.-K/s   in 0s

2017-09-12 12:00:00 (36.2 MB/s) - ‘index.html’ saved [1270/1270]

user@remote:~/scits-training/ $ tail index.html
```

Mnemonic:

`wget` stands for **Web get**

Downloading from the shell

Notable option: renaming the file immediately.

- **-O** (for **o**utput) chooses a specific file to write to

```
$ wget https://tools.ietf.org/rfc/rfc1149.txt -O april.txt
[...]
$ less april.txt
```

As usual, use `man wget` to see more options.

It can work with HTTP/HTTPS/FTP-hosted files.

Transferring files between systems

To send files between two computers using SSH, the simplest command is `scp`.

Mnemonic:

`scp` stands for **secure copy**

`scp` behaves a lot like `cp`, but you can provide locations on other computers.

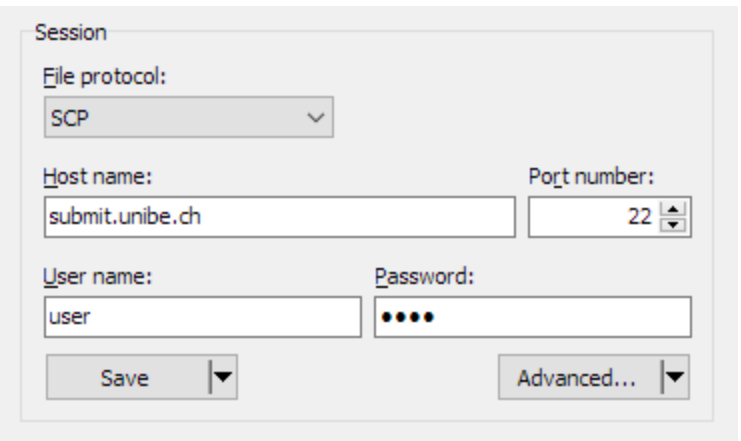
How to use `scp` on your own machine depends on the OS.

`scp` on Windows

While PuTTY includes a command-line client `pscp` with the same functions, it may be better to use a GUI client `WinSCP`.

It can be downloaded from https://winscp.net/

You can then connect using SCP (or SFTP) with your normal credentials and transfer files between your PC and the remote:



`scp` on Linux / MacOS

From your **local** terminal, you can transfer a file from UBELIX:

```
user@local:~ $ scp user@submit.unibe.ch:~/scits-training/numbers/hundred .
[..some authentication..]
user@local:~ $ less hundred
```

`scp`'s parameters work similarly to `cp`, but you can refer to files on other systems by adding `user@remote:` to the path.

It works both ways, and can rename as well:

```
$ scp hundred user@submit.unibe.ch:~/scits-training/numbers/another_hundred
```

Moving data in and out

Exercise:

1. Copy all of the `B`-files from `moving/source/subfolder` to your computer with one command (use wildcards).
2. Copy some folder from your computer to the home folder of the remote system (use `-r`).

Moving data in and out

Exercise:

1. Copy all of the `B`-files from `moving/source/subfolder` to your computer with one command (use wildcards).
2. Copy some folder from your computer to the home folder of the remote system (use `-r`).

In addition to `scp`, there's a command that works better for repeatedly copying large folders with small changes: **`rsync`**.

It will not be covered here, but look up information on it if it's your use case.