

Python Programming for Data Science

WEEK 43, MONDAY

WORKING ON REMOTE SERVERS:

Running on remote servers

Basic Unix commands

Compressing and transferring files

Remote servers

When analyzing real-life datasets, you will often work on remote servers.

Why?

1. When processing large files, you might need more compute-power than what your laptop provides
2. Sensitive data might not be allowed to leave the server on which it is stored (e.g. registry data)
3. You might want to run computations for several days, making it inconvenient to use your laptop.

Remote servers often run Linux/Unix. We'll therefore spend some time on how to work on such a server.

VPN - Virtual Private Network (1)

For security reasons, the University of Copenhagen requires access to its servers to go through VPN - using multi-factor-authentication

Instructions for how to set this up can be found here:

- Windows: <https://kunet.ku.dk/employee-guide/ITvejl/VPN%20installation%20on%20Windows%20for%20students.pdf>
- Mac: <https://kunet.ku.dk/employee-guide/ITvejl/VPN%20installation%20on%20Mac%20-%20for%20students.pdf>
- Linux: <https://kunet.ku.dk/employee-guide/ITvejl/VPN%20installation%20on%20Linux%20-%20for%20students.pdf>

VPN - Virtual Private Network (2)

After having installed Cisco Anyconnect, you should connect using the following steps

1. Open Cisco Anyconnect
2. Connect to `vpn.ku.dk`, and fill in your KU credentials



3. Cisco Anyconnect will now wait... until you have approved the log-in using NetIQ on your phone.
4. After approving, the VPN connection will be established.

Logging in to a remote Unix server

Logging in to a remote server using ssh

On MacOS and Linux/Unix, there is a built-in terminal (called terminal or console), from which you can use the command `ssh` to log into a remote server:

```
ssh user-name@server-name
```

Example:

```
$ ssh knv868@ppds.unicph.domain      # user knv868
Password:                                # I am now on the remote server
$
```

Logging in to a remote server on newer versions of Windows

Recent versions of Windows have ssh available by default. To start ssh, go to the start menu, and type Command Prompt or Powershell. From there, you can type the ssh command at the prompt:

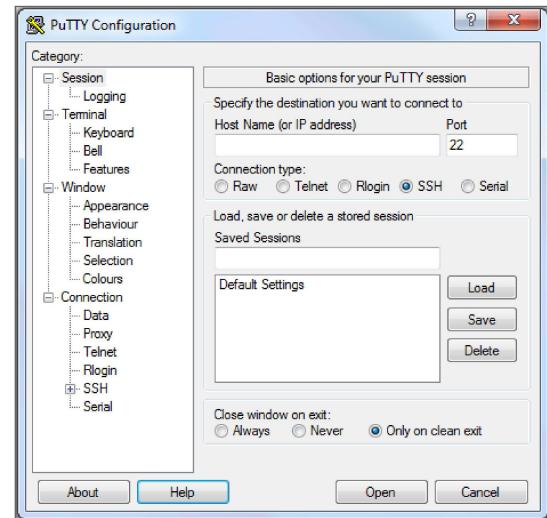
```
ssh user-name@server-name
```

If the ssh command does not work, you might have to enable it. See [this guide](#) for details.

Logging in to a remote server on Windows

If you cannot get the built-in ssh version in Windows to work, you can also use the Putty ssh-client, which is extremely easy to install:

1. Downloads Putty from
<https://www.putty.org/>
2. Start the program, and fill the server name into the Host Name field. Make sure that SSH is selected.



Remote servers in this course

To experiment with logging in to a remote server, we have made a server available. The server name is `ppds.unicph.domain.`

Remember, in order to access the above server, you will need to connect to your VPN (`vpn.ku.dk`) - for instance using Cisco Anyconnect.

Exercise - Logging into a remote server

1. Start your VPN.
2. Try to log in to the ppds.unicph.domain server using either Putty or the ssh command.
3. You can exit again by writing exit, or pressing ctrl-d.

Basic Unix commands

The shell

Within the terminal window, there is a command-line interpreter, typically referred to as the **shell**.

```
lpp@lpp-vbox:~$ █
```

Any command that you type here will be executed when you press enter.

There are several available shells: tcsh, bash, zsh, ... Their core functionality is the same.

In this course, we will use **bash (bourne again shell)**.

Command-line interfaces — why?

Command line interfaces seem old-fashioned. Why are they useful?

A command line interface has many advantages:

- More control
- No mouse needed
- Easier to automate tasks (scripting)
- Easier to run tasks on a remote server

The Unix/Linux command line is quite powerful
(as we will see)

Commands

Executing a command: `cal`

```
$ cal
September 2020
Su Mo Tu We Th Fr Sa
      1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

Specifying options to a command

Specifying options to a command:

```
$ cal -m
September 2020
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

Which options are available?

Two strategies:

- Google it
- Use the `man` program

```
$ man cal
CAL(1)                               User Commands                  CAL(1)

NAME
    cal - display a calendar

SYNOPSIS
    cal [options] [[[day] month] year]

DESCRIPTION
    cal displays a simple calendar. If no arguments are specified, the
    current month is displayed.

OPTIONS
...
    -m, --monday
        Display Monday as the first day of the week.
...
```

Use the "q" key to quit the manual viewer.

Python from the command line

You can run Python from the command line. Just typing `python` gives you an interactive Python session:

```
$ python
Python 3.8.0 (default, Apr 28 2020, 13:27:51)
[GCC 9.3.1 20200408 (Red Hat 9.3.1-2)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

You can also execute a `.py` file:

```
$ python my_script.py
Hello, World!
```

Shell: history

Avoid typing the same commands again

Different techniques:

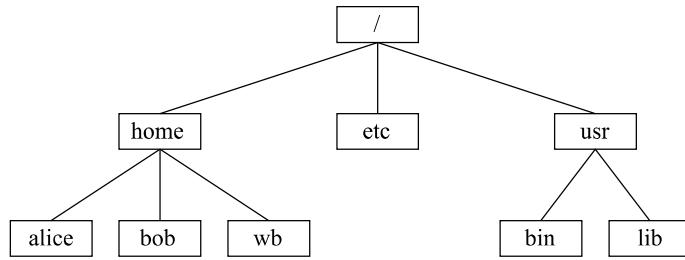
- The **history** command
- Pressing the up (and down) arrow keys.
- *Ctrl-r* : Searches backwards through history

```
$ history  
...  
3  cal  
4  cal -m  
5  man cal  
6  history
```

The shell: miscellaneous

- Cancel a program running in the shell: *Ctrl-c*
- Clear the screen: *Ctrl-l*
- Close the terminal window: *Ctrl-d* (when the prompt is empty)

Remember the directory structure?

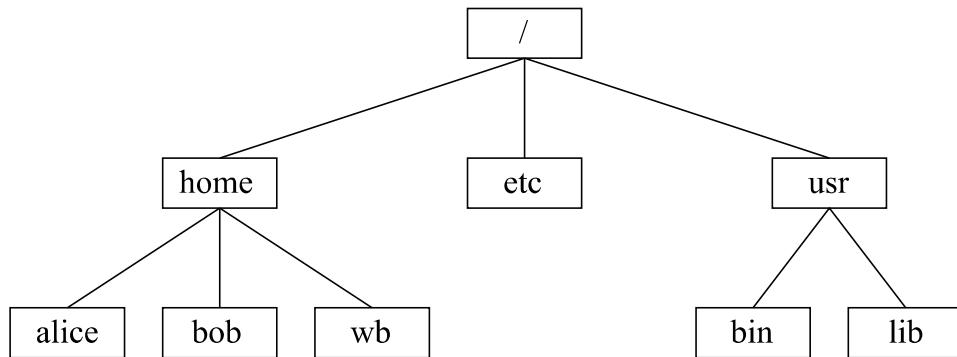


- The *current directory* is where you are in the tree right now.
- You can use the **pwd** command to get the current directory (*print working directory*):

```
[knv868@ppdsapp01f1 ~]$ pwd  
/home
```

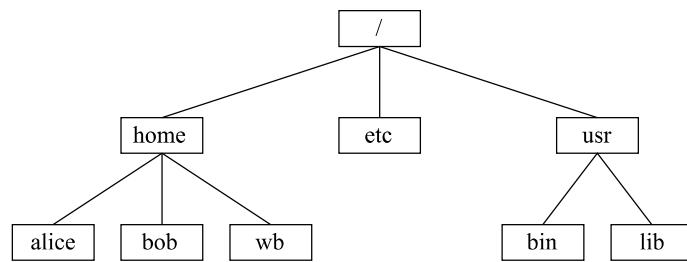
- This information is also displayed in your prompt.

Reminder: Root directory



- The root directory is the directory at the highest level of the file system hierarchy.
- On unix systems, it is referred to by /

Reminder: Absolute and Relative paths



Directories (folders) and files can be referred to by using either an **absolute** or a **relative** path:

absolute:

/home/wb/exam_results.html

relative: (assuming I am in /home)

wb/exam_results.html

Special path names

. Current directory

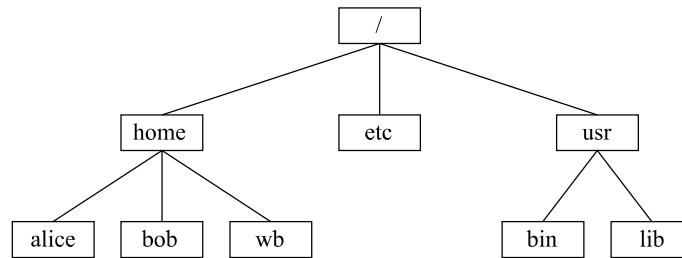
.. Parent directory

~ Home directory

~user Users home directory

Navigating the directory tree

Use the `cd` command to change directory

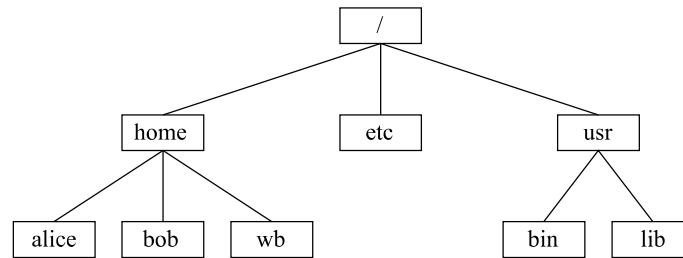


Examples:

```
~$ cd /etc          # absolute path name
/etc$ cd ..         # relative path name: go to parent
/$ cd              # cd without argument is the same as cd ~
(home dir)
~$                  # back in home directory
```

Listing files

The **ls** command lists files and directories



Examples:

```
~$ ls          # list files/directories from current directory
$ ls /home/alice # list files/directories from alice's directory
$ ls ..        # list files/directories from parent (/home)
```

Listing files — options

The **ls** command takes several options.

For example:

-l Long format (more details)

-a All files (including files starting with a dot)

-F Append a slash to directory names

```
~$ ls -laF
-rw-----. 1 knv868 users 5 Oct 19 16:10 .bash_history
-rw-----. 1 knv868 users 18 Oct 19 16:09 .bash_logout
-rw-----. 1 knv868 users 141 Oct 19 16:09 .bash_profile
```

Files starting with a dot are typically hidden (unless you use the -a option)

Unix commands - Exercise 1

- Log into the server
- Check what directory you are in
- Go to the parent directory
- Get a list of all the files in this directory - What do you see?
- Come up with three different ways to come back to your home directory
- Use **ls** to see the all the files in your home directory
- Use **ls** to see the all the files in your home directory sorted by date

Unix commands - Exercise 1 — solution

```
~$ pwd          # check which directory I'm in
```

```
~$ cd ..        # Go to parent directory
```

```
unicph.domain$ ls          # List of all files  
ls: cannot open directory '.': Permission denied
```

```
unicph.domain$ cd          # Jump to home directory  
unicph.domain$ cd ~         # Jump to home directory  
unicph.domain$ cd knv868    # Jump to home directory  
(relative path)
```

```
~$ ls -la          # lists all files (including hidden) -  
long format
```

```
~$ ls -lat         # lists all files sorted by date - long  
format
```

Viewing files

Reading a file called *file-name* using the **less** command:

```
less file-name
```

Important keys in less:

space page down

b page up

/ search

q quit

There is a command called **more** which does basically the same (but only allows you to go forward)

Viewing files — example

```
~$ less .bash_history      # Where the shell stores your  
history  
cal  
cal -m  
man cal  
...
```

Creating directories

Create a new directory: `mkdir`

```
mkdir directory
```

Example:

```
~$ mkdir ppds_handins  
~$ ls -lt      # Check by listing with newest first  
(long format)
```

File & directory names

Most characters are allowed. Beware of spaces since they are used to separate commands and arguments:

```
~$ less ppds exercise one.txt  
ppds: No such file or directory  
exercise: No such file or directory  
one.txt: No such file or directory
```

Recommendation: Use only letters, numbers, underscore(_), dash(-) and dot(.). Avoid æ, ø, å, œ, ç ...

File & directory names (2)

Dealing with space-containing filenames

Solution: *quote* text:

```
~$ less "ppds exercise one.txt"
```

or escape problematic characters:

```
~$ less ppds\ exercise\ one.txt
```

Auto-completion

The shell can guess which file or directory you are interested in. This is done using the TAB key:

```
~$ mkdir ppds_material
~$ mkdir ppds_exercises
~$ cd ppds_m          # TAB
~$ cd ppds_material
```

This allows you to type very long commands in a short time.

Auto-completion (2)

If there is more than one option, the shell completes as far as it can and then lists the possibilities for completion.

```
~$ cd p          # TAB
~$ cd ppds_      # TAB (or TAB TAB)
ppds_exercises/ ppds_material/
~$ cd ppds_m     # TAB
~$ cd ppds_material
```

In some shells you will have to press TAB twice before the list shows up.

Wildcards

Used to specify a selection of files:

Three types:

* Match any character any number of times

? Match any character once

[] Match specified character class once

```
~$ mkdir exercise1 exercise2 exercise20      # Make 3  
directories  
~$ ls -d exercise*  
exercise1 exercise2  exercise20  
~$ ls -d exercise?  
exercise1 exercise2  
~$ ls -d exercise[2-4]  
exercise2
```

Note that these are different than those used in regular expressions (for instance: ".*" vs "*").

Wildcards (2)

Examples of character classes to be used in []:

a-z a to z, small letters

A-Z A to Z, capital letters

a-zA-Z all letters, lower and upper case

0-9 0 to 9

0-9,a-zA-Z 0 to 9 and a to z

Copying files (`cp`)

Two ways to copy a file:

1. Copy file called *original* to file called *copy*

```
cp original copy
```

2. Copy file called *original* and put it in *target-directory*

```
cp original target-directory/
```

Examples:

```
~$ cp .bash_history .bash_history_backup #1. Copy to new  
filename  
~$ mkdir backup  
~$ cp .bash_history backup/ #2. Copy into  
directory
```

Copying files (`cp`) - overwriting

Note that if the target name already exists, it will be replaced without warning!

```
~$ cp .bash_history bla.txt #1 Overwrites bla.txt if it exists
```

Moving/renaming files

Two ways to move/rename a file: `mv`

1. Rename file called *original* to file called *new*

```
mv original new
```

2. Move file called *original* and put it in *target-directory*

```
mv original target-directory
```

Examples:

```
~$ mv .bash_history_backup .bash_history_old    #1. Rename  
file  
~$ mkdir old_stuff/  
~$ mv .bash_history_old old_stuff/               #2. Move into  
directory
```

Removing/deleting files

Remove a file: `rm`

```
rm filename
```

Note: In the shell there is no trash can or recycle bin where files are stored when they are deleted. Files are deleted directly and **there is no way to undo a delete.**

```
~$ rm .bash_history_backup          # Gone forever
```

Removing directories

remove empty directory

```
rm -r directory/
```

remove non empty directory and all subfolders/files within

```
rm -rf directory/
```

Q: What would happen if one typed? (**Do not do this!**)

```
rm -rf /
```

A: you would delete all your files on the system

Creating symbolic links

Create link called *link-file* that points to a file *original*:

```
ln -s original link-file
```

Exactly like aliases (in Mac) or shortcuts (in Windows). Useful when dealing with large datasets, where making copies is expensive.

Example:

```
ln -s /usr/share/dict/british-english words_uk
```

Transferring files between systems (scp)

To copy files between two systems (e.g. from a remote server and your home computer), use `scp` from remote to you:

```
scp user-name@server:remote-path local-path
```

from you to remote:

```
scp local-path user-name@server:remote-path
```

Example:

```
scp /usr/share/dict/british-english  
knv868@ppds.unicph.domain:ppds_material  
# ppds_material is relative to your home-directory
```

Like for `cp`, there is a `-r` option for copying directories

Unix commands - Exercise 2

- Create a directory called **temp**
- Try to move the English word list (previous slide) to this directory
- Try to copy the English word list (previous slide) to this directory
- Try to make a symbolic link from the English word list (previous slide) to this directory
- Remove the temp directory and its content.

Unix commands - Exercise 2 — solution

- Create a directory called **temp**

```
~$ mkdir temp/      # Create a directory
```

- Try to move the English word list (previous slide) to this directory

```
~$ cd temp          # Make temp directory current
```

```
~/temp$ mv /usr/share/dict/british-english .    #
Attempt to move file
mv: cannot move '/usr/share/dict/british-english' to
'./british-english':
Permission denied
```

- Try to copy the English word list to this directory

```
~/temp$ cp /usr/share/dict/british-english .      #
Copying instead
```

Unix commands - Exercise 2 — solution (2)

- Try to make a symbolic link from the English word list to this directory

```
~/temp$ rm british-english          # Delete  
local copy
```

```
~/temp$ ln -s /usr/share/dict/british-english . # Make  
symlink
```

- Remove the temp directory

```
~/temp$ cd ..      # Change current directory to home  
directory
```

```
~$ rm -rf temp    # Remove temp directory and all its  
contents
```

More Unix commands: viewing and searching in files

Unix commands: concatenating files

Concatenate and output files with **cat**

```
cat file
```

Often, this program is used to simply output a single file to the screen:

```
~$ cat .bash_history
cal
cal -m
man cal
...
```

Note that **cat** doesn't pause at each page like **more** or **less**

Unix commands: output simple text

The **echo** command outputs text to screen

```
echo "text"
```

Example:

```
~$ echo "Hello World!"  
Hello World!
```

This might not seem terribly useful, but it is handy for inspecting environment variables:

```
~$ echo $PATH  
/home/unicph.domain/knv868/.local/bin:/home/unicph.domain/  
knv868/bin...
```

This is useful when you want to configure your shell

Unix commands: create an empty file

The **touch** command is a quick way to change a file's timestamp (date of latest modification)
It is also a easy way to create an empty file:

```
touch filename
```

Example:

```
~$ touch my_test_file.txt      # create file
~$ ls -l my_test_file.txt    # verify that it has been
                             created
-rw-rw-r-- 1 lpp lpp 0 Aug 31 11:11 my_test_file.txt
```

Unix commands: Beginning and end

To see only the first n lines of a file, use **head**:

```
head filename # By default, n=10 if not specified
```

Similarly, **tail** shows the last n lines:

```
tail filename # By default, n=10 if not specified
```

Example:

```
~$ head -n 3 /usr/share/dict/british-english    # First 3 lines
A
A's
AA's
~$ tail -n 3 /usr/share/dict/british-english    # Last 3 lines
étude
étude's
études
```

Unix commands: counting in a file

The **wc** command counts lines, words and characters in a file

```
wc filename
```

Example:

```
~$ wc /usr/share/dict/british-english  
101825 101825 967402 /usr/share/dict/british-english #  
lines, words, bytes
```

Unix commands: sorting

Use the sort to sort the contents of a file

```
sort filename
```

Example:

```
~$ sort /usr/share/dict/british-english
a
A
Aachen
...
Zyrtec's
Zyuganov
Zyuganov's
```

Unix commands 3 - Exercise

- Figure out how to get `wc` to output *only* the number of lines
- What is the 123th line of the dictionary file from the previous slide?
- Sort the dictionary in reverse order

Unix commands 3 - Exercise — solution

- Figure out how to get `wc` to output *only* the number of lines

```
~$ wc -l /usr/share/dict/british-english  
101825
```

- What is the 123th line of the dictionary file from the previous slide?

```
~$ head -n 123 /usr/share/dict/british-english  
...  
Adhara'sa  
Adidas  
Adidas's
```

- Sort the dictionary in reverse order

```
~$ sort -r /usr/share/dict/british-english
```

Unix commands: searching within files

Search for a regular expression inside one or more files:
grep

```
grep pattern filename
```

grep will return all lines that match your query.

```
~$ grep "^[Aa]pple" /usr/share/dict/british-english
Apple
Apple's
Appleseed
...
```

Remember that you can also use it to search for normal search strings

```
~$ grep "apple" /usr/share/dict/british-english
Mapplethorpe
Snapple
apple
...
```

Unix commands: searching within directories

You can tell **grep** to search through all files in a directory, and all its subdirectories, *i.e.* recursively

```
grep -r pattern directories
```

Example:

```
~$ grep -r apple /usr/share/dict/
/usr/share/dict/british-english:Mapplethorpe
/usr/share/dict/british-english:Mapplethorpe's
...
/usr/share/dict/american-english:Mapplethorpe
/usr/share/dict/american-english:Mapplethorpe's
...
```

Unix commands: searching for files

Use the **find** command to search for files by their name:

```
find start-directory -name pattern
```

You can also use **find** to search for other properties of the file (modification time etc)

Example:

```
~$ find /usr/share -name "brit*" # Search for files
starting with "brit"
/usr/share/dict/british-english
/usr/share/man/man5/british-english.5.gz
~$ find .                                         # Lists all files in
current directory.
```

Note that the "patterns" used by **grep** are not the same as those used by **find** (**grep** uses *regular expressions*)

Unix commands: search-replace within files

Using **sed** (stream line editor), you can do search-replace

```
sed 's/query/replacement/g' filename
```

s = substitute and **g** = global

sed has lots of other features, but this is all we'll cover here.

Example:

```
~$ sed 's/apple/pear/g' /usr/share/dict/british-english      #
apple -> pear
...
Mpearthorpe's
...
```

Getting help

As mentioned, `man` gives a description of any installed program:

```
man program-name
```

`man` uses `less` to display the help pages. For convenience, we repeat the list of important keys in `less`:

space	page down	<i>b</i>	page up
/	search	<i>q</i>	quit

Remember that Google is a useful resource as well.

Getting help (2)

If you just want a short list of possible options (parameters), try

```
program-name --help
```

Example:

```
~$ ls --help
Usage: ls [OPTION]... [FILE]...
...
-a, --all                         do not ignore entries
starting with .
...
```

Sometimes, depending on the program, the way to call its help can vary: **-help**, **--help**, **-h**, **--h**

Getting help: man vs help

There is no man-page for the **cd** command. Why?

Answer: **cd** is a command that is built-in to the shell, and these do not have man-pages

Solution: Use **help** instead.

```
~$ help cd
cd: cd [-L|[-P [-e]] [-@]] [dir]
      Change the shell working directory.
...
```

Unix commands 4 - Exercise

- Create a directory called `dict_exercise`
- From the man-page for the `cp` command, find out how to copy a whole directory (hint: recursively)
- Copy the `/usr/share/dict` directory to the `dict_exercise/` directory.
- Find all occurrences of `apple` in the `british-english` file, but using a case-insensitive search (matching both `apple` and `APPLE`).
- Replace all occurrences of the word `circle` with `square` in the `british-english` dictionary.

Unix commands 4 - Exercise — solution

- Create a directory called dict_exercise

```
~$ mkdir dict_exercise
```

- From the man-page for the cp command, find out how to copy a whole directory (hint: recursively)
- Copy the /usr/share/dict directory to the dict_exercise/ directory.

```
~$ cp -r /usr/share/dict/ dict_exercise/
```

Unix commands 4 - Exercise — solution (2)

- Find all occurrences of apple in the british-english file, but using a case-insensitive search (matching both Apple and apple).

```
~$ grep -i apple dict_exercise/dict/british-english
```

- Replace all occurrences of the word circle with square in your copy of the british-english dictionary.

```
~$ sed 's/circle/square/g' dict_exercise/dict/british-english
```

or

```
~$ sed 's/circle/square/g;s/Circle/Square/g'  
dict_exercise/dict/british-english
```

Compressing files

Why compress?

We saw earlier how to transfer files between servers. Since bandwidth costs money, it makes sense to compress file content as much as possible. This is why many of the downloadable files on the internet are distributed as e.g. `.gz`, `.bz2` or `.zip`.

zip

Compressing a file using zip:

```
zip name-of-new-zip-file filenames-to-compress
```

Example:

```
~$ zip british-english.zip /usr/share/dict/british-english
```

To zip an entire directory, use the -r flag:

```
~$ zip -r zip -r handin1.zip handin1/
```

gzip

Compressing a file using gzip:

```
gzip filename
```

Note that this will *replace* your old file with the gzipped equivalent (.gz)

Example:

```
~$ cp /usr/share/dict/british-english .
~$ gzip british-english          # Creates british-
english.gz
```

bzip2

Compressing a file using bzip2:

```
bzip2 filename
```

Note that this will *replace* your old file with the bzip2ed equivalent (.bz2)

Example:

```
~$ cp /usr/share/dict/british-english .
~$ bzip2 british-english          # Creates
british-english.bz2
```

Wrapping things up with tar

gzip and bzip2 work on single files

If you want to compress a lot of files, you can use tar to wrap them into a single file, and then compress this file.

```
tar cvf name-of-new-file.tar files-or-directories
gzip name-of-new-file.tar          # Produces tar.gz file
```

This combination of commands is so common, that you can ask tar to do both:

```
tar -zcvf name-of-new-file.tar.gz files-or-directories      #
produces tar.gz
tar -jcvf name-of-new-file.tar.bz2 files-or-directories      #
produces tar.bz2
```

Example:

```
~$ tar -zcvf dictionaries.tar.gz /usr/share/dict      #
produces .tar.gz
```

Uncompressing

You can uncompress with the corresponding un-program:

```
unzip filename.zip          # creates filename from  
filename.zip                # replaces filename.gz with  
gunzip filename.gz          # replaces filename.bz2 with  
filename                  # replaces filename.bz2 with  
bunzip2 filename.bz2        # replaces filename
```

Unpacking a tar-file is done using the `-x` flag:

```
tar -xvf filename.tar
```

This also works on tar.gz or tar.bz2 files:

```
tar -xvf filename.tar.gz      # First uncompresses, then  
unpacks                      # First uncompresses, then  
tar -xvf filename.tar.bz2      # First uncompresses, then  
unpacks
```

Example

```
~$ tar -xvf dictionaries.tar.gz
```

Compression - Exercise

Let's try to compress the /usr/share/dict/british-english file

1. Copy the file to a file in your current directory called "mydict.txt"
2. Compress it using zip.
3. Compress it using gzip.
4. Compress it using bzip2.
5. Compare the file sizes of the outputs produced by the three methods

Compression - Exercise — solutions

1. Copy the file to a file in your current directory called "mydict.txt"

```
~$ cp /usr/share/dict/british-english mydict.txt
```

2. Compress it using zip.

```
~$ zip mydict.txt.zip mydict.txt  
adding: mydict.txt (deflated 73%)
```

3. Compress it using gzip.

```
~$ gzip -k mydict.txt # Compress  
mydict.txt into mydict.txt.gz
```

4. Compress it using bzip2.

```
~$ bzip2 -k mydict.txt # -k keeps the original  
file
```

Compression - Exercise — solutions (2)

5. Compare the file sizes of the outputs produced by the three methods

```
~$ ls -l mydict.txt*
-rw-r--r-- 1 knv868 sec-sci-users 967402 Oct 23 20:58
mydict.txt
-rw-r--r-- 1 knv868 sec-sci-users 344251 Oct 23 20:58
mydict.txt.bz2
-rw-r--r-- 1 knv868 sec-sci-users 259019 Oct 23 20:56
mydict.txt.gz
-rw-r--r-- 1 knv868 sec-sci-users 259160 Oct 23 20:56
mydict.txt.zip
```

Surprisingly, bz2 seems to perform the worst on this particular dataset (although it is generally considered the best algorithm of the three).

Speaker notes