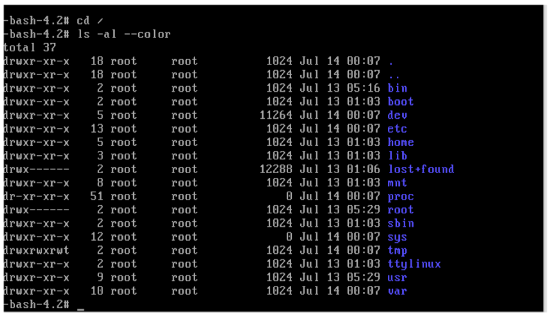
DR TIM KAHLKE
Tim.Kahlke@uts.edu.au

Bioinformatics

Introduction

Graphical vs command line interface

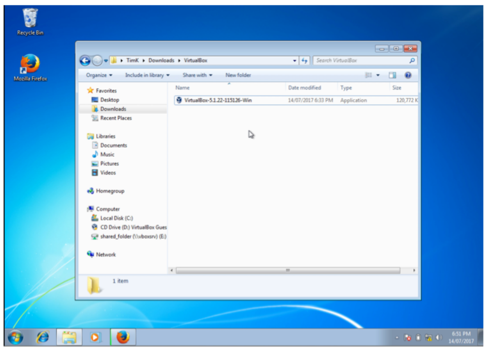
The *command line interface* (CLI) often simply referred to as *the command-line*, *console* or *terminal*, is a way of interacting with a computer by means of typing *commands* on a specific *line* and submitting them to the computer by hitting the *enter* key. Historically, the command-line was the initial way of interacting with a personal computer: a user typed in a commands using the keyboard and the computer executed them.



```
-bash-4.2# cd /
-bash-4.2# ls -al --color
total 37
drwxr-xr-x   18 root     root          1024 Jul 14 00:07 .
drwxr-xr-x   18 root     root          1024 Jul 14 00:07 ..
drwxr-xr-x    2 root     root          1024 Jul 13 05:16 bin
drwxr-xr-x    2 root     root          1024 Jul 13 01:03 boot
drwxr-xr-x    5 root     root         11264 Jul 14 00:07 dev
drwxr-xr-x   13 root     root          1024 Jul 14 00:07 etc
drwxr-xr-x    5 root     root          1024 Jul 13 01:03 home
drwxr-xr-x    3 root     root          1024 Jul 13 01:03 lib
drwx------    2 root     root         12288 Jul 13 01:06 lost+found
drwxr-xr-x    8 root     root          1024 Jul 13 01:03 mnt
dr-xr-xr-x   51 root     root             0 Jul 14 00:07 proc
drwx------    2 root     root          1024 Jul 13 05:29 root
drwxr-xr-x    2 root     root          1024 Jul 13 01:03 sbin
drwxr-xr-x   12 root     root             0 Jul 14 00:07 sys
drwxrwxrwt    2 root     root          1024 Jul 14 00:07 tmp
drwxr-xr-x    2 root     root          1024 Jul 13 01:03 ttylinux
drwxr-xr-x    9 root     root          1024 Jul 13 05:29 usr
drwxr-xr-x   10 root     root          1024 Jul 14 00:07 var
-bash-4.2# _
```

Example CLI
Command-Line Interface with listed directory content

In contrast to the *command line interface* we interact with modern computers through *graphical user interfaces* (GUIs)*:* on start-up we are greeted with logos of the specific computer vendor, colourful backgrounds and icons for the different programs, files and folders which we can click or tab on using a *mouse-pointer* or touch screen. These GUIs, i.e., graphical representations of the computer content, clickable icons and a connected graphical *mouse-pointer,* were crucial for the success of the personal computer.



Example GUI
Windows Desktop with open Control Panel

Introduction

Why still use the command-line?

Despite their obvious benefits GUIs have certain disadvantages: modern operating systems, e.g. Windows and Mac OSX, increasingly “hide” information from the user to make software more user-friendly as well as to protect their business model. Examples are the so called “hidden files” and “file extensions” that by default are not shown to the user unless these options are specifically activated in Windows and Mac OSX settings. However, these files are important, e.g. for systems administrators and developers, that often need excess to all features and files of a computer system. The command line can give full access to the complete computer and is therefore the interface of choice for many people working in IT.
Another important reason for using the command line is that it simplifies software development: developing a GUI, i.e., buttons and windows for your app can be a very time consuming process including graphic design, the development itself and testing. Writing command line scripts, however, reduces the development to the core functionality of the tool. Additionally, in contrast to most GUI programs, many command line scripts can be executed on many different operating systems: they are *cross-platform compatible*.

Command line and bioinformatics

Many tools and programs in bioinformatics are command-line based programs that either don’t need or simply don’t have a GUI. In fact for a lot of bioinformatics application cases a GUI is more of a detriment than a benefit. A significant part of bioinformatics consists of parsing, converting or otherwise manipulating large text-based data files. In the fast evolving field of bioinformatics with its frequentliy changing file formats and short half-life of existing tools keeping GUI software up to date is almost impossible. Additionally, many bioinformatics tools are written by researchers from many different research labs and the broader bioinformatics community and not by professional software companies.

Introduction

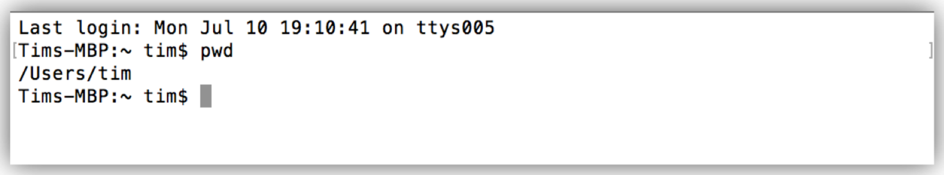
Light-weight command line based tools that can easily be shared, changed and adapted to new tasks are therefore the first choice of most bioinformatics developers. Although a few GUI based bioinformatics software frameworks exists that provide general workflows for bioinformatics analysis the vast majority of bioinformatics tools are command line based and new developments in the field are almost exclusively provided in form of scripts and packages written in python, R, perl or other language.

Command line - Navigation

Location, location, location …

When first working with a command line it is useful to keep in mind that a command line and a GUI have the same purpose: show and interact with the different programs, files and directories on a computer. The difference is simply how they do it. For example, when you log in on a Windows computer you see the Desktop and all the folders and files that are on the it. In contrast, when you start a command line you see a more or less empty window. To get the current location, i.e., the directory you are in, and to show the that directory one has to type specific commands on the command line.

Let's start with the location. To see in which directory we are at the moment we can type the command ***pwd*** (short for *print working directory*) which prints out the current location:



```
Last login: Mon Jul 10 19:10:41 on ttys005
Tims-MBP:~ tim$ pwd
/Users/tim
Tims-MBP:~ tim$ 
```

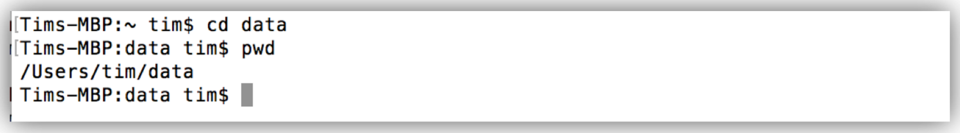
As you can see in the example above the so called ***path*** to the current working directory is */Users/tim*. The '/' symbol in a path denote a directories: thus */Users/tim* indicates that we are in directory /*tim* which is a subdirectory of directory /*Users.*

Command line

Your *home* directory

The default directory a user starts in after the login is also called the ***home directory*** of that user. It is usually named after the user (hence the */tim* directory). On Apple systems all home directories are found in the /Users directory. On Unix-based systems such as Ubuntu the home directories are usually located in */homes*.

When we found out where we are we want to know how to change into another directory. For example, let's say we know that there is a directory *data* in our home directory. To change into the *data* directory we use the command ***cd*** (short for *change directory*) followed by the location of the directory we want to change into.

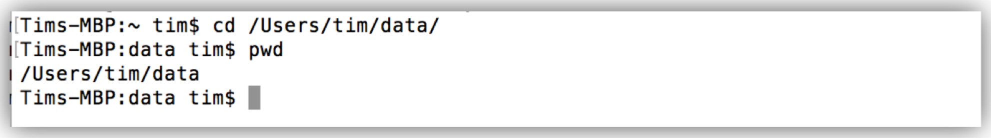


```
Tims-MBP:~ tim$ cd data
Tims-MBP:data tim$ pwd
/Users/tim/data
Tims-MBP:data tim$ 
```

When we use the ***pwd*** command to check that we actually changed into the */data* directory the output shows */Users/tim/data* as our new location. This is the ***absolute pat*h** of the directory *data*. In contrast we only used the ***relative path*** for the *cd* command. ***Relative paths*** are given *relative to the current working directory*, i.e., the directory you are currently in. Additionally, relative paths do NOT start with a "/" but either without a slash or using the notation "./" which means "this directory".

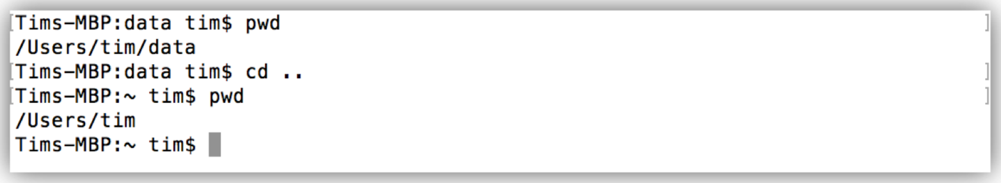
Command line

In contrast to relative paths ***absolute paths*** specify a directory *regardless of the current working directory*. This means that we can change into the *data* directory from anywhere by using its absolute path. Absolute paths ALWAYS start with a “/” as they denote the the path from the lowest directory, i.e. *root* directory, down to the specified location:



```
Tims-MBP:~ tim$ cd /Users/tim/data/
Tims-MBP:data tim$ pwd
/Users/tim/data
Tims-MBP:data tim$ 
```

The last command for navigation on the command line is used to go one directory *up* into the *parent directory* of our current location. On the command line the *parent directory* of any location is specified by two dots. This means if we are still in */Users/tim/data* and want to change into the parent directory */Users/tim* we can do so by typing ***cd ..***



```
Tims-MBP:data tim$ pwd
/Users/tim/data
Tims-MBP:data tim$ cd ..
Tims-MBP:~ tim$ pwd
/Users/tim
Tims-MBP:~ tim$ 
```

Self Check 1

1. **What does CLI stand for?**

2. **How do you find out what your current location is?**

3. **How do you change into another directory?**

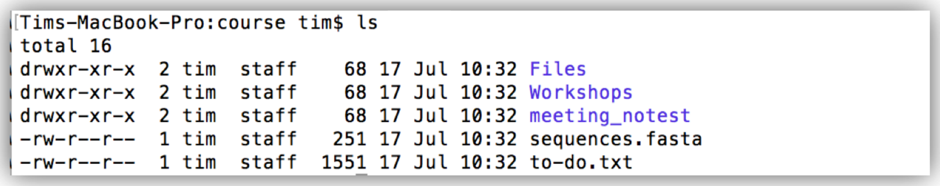
4. **Can you think of way to change into the parent directory other than using the command '*cd ..*' ?**

Command line

Where are my files ?

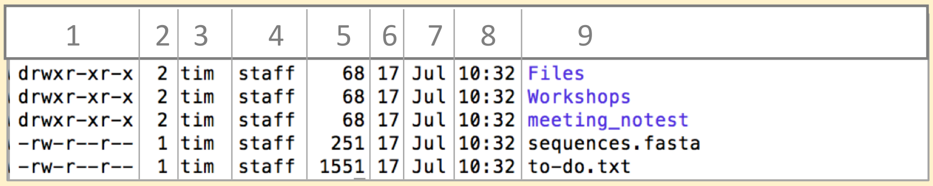
Now that we are able to navigate on the command line the next question is: how do I find out what files are in my directory?

To view the content of a directory use the command ***ls*** (short for ***list***) which shows one line per file, program or directory in the specified location:



```
Tims-MacBook-Pro:course tim$ ls
total 16
drwxr-xr-x  2 tim  staff    68 17 Jul 10:32 Files
drwxr-xr-x  2 tim  staff    68 17 Jul 10:32 Workshops
drwxr-xr-x  2 tim  staff    68 17 Jul 10:32 meeting_notest
-rw-r--r--  1 tim  staff   251 17 Jul 10:32 sequences.fasta
-rw-r--r--  1 tim  staff  1551 17 Jul 10:32 to-do.txt
```

Wow, ok, we can see that there are a kit if things in there. Let's take a closer look: first of all we see that the output of *ls* consists of nine columns:



1	2	3	4	5	6	7	8	9
drwxr-xr-x	2	tim	staff	68	17	Jul	10:32	Files
drwxr-xr-x	2	tim	staff	68	17	Jul	10:32	Workshops
drwxr-xr-x	2	tim	staff	68	17	Jul	10:32	meeting_notest
-rw-r--r--	1	tim	staff	251	17	Jul	10:32	sequences.fasta
-rw-r--r--	1	tim	staff	1551	17	Jul	10:32	to-do.txt

<u>Explanation of tabular *ls* output</u>

1. Access permissions, who is allowed to **r**ead, **w**rite and e**x**ecute the file
2. How many links it has
3. Who owns it
4. What group it is in
5. The file size in bytes
6. Last modified (day)
7. Last modified (month)
8. Last modified (time or year)
9. Name of the file/directory

Command line

The most important columns for us at the moment are the columns 3, 5 and 9:

3 - who created the file

5 - how big it is

9 - the name of the file/directory.

Additionally, the *ls* output distinguishes between files and directories: directory names are marked in blue and file names are black. Another way of telling the difference between a file and a directory is the first column: directories have a small *'d'* as the first character in column 1 where files only have a dash.

With this knowledge we can now properly interpret the output of *ls* of our current working directory:

- It contains 3 directories: Files, Workshops and meeting_notes
- Additionally It contains 2 files: sequences.fasta and to-do.txt
- All files were created by user me (user *tim)*
- The file *sequences.fasta* is 251 bytes and file *to-do.txt* is of size 1551 bytes.

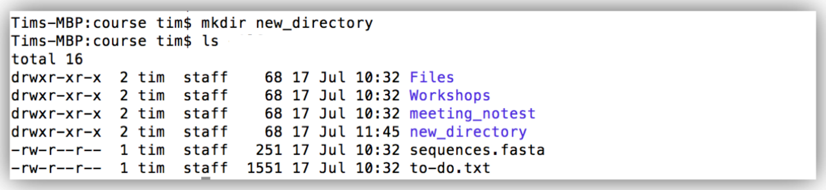
Note on *ls*

On many command line systems the native *ls* command will only show the names of the files and directories in one line without additional information. To show the output shown here you will have to add additional *arguments* to the *ls* command. However, for the course and the exercise you will not have to add these arguments and the output above will be the default output of the *ls* command!

Command line - Commands

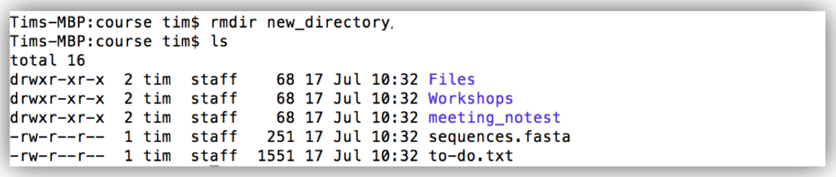
Options, arguments and parameters

We have already used several different command line commands, e.g. *pwd* and *cd*. The difference between these two commands is that *pwd* can be run by itself whereas *cd* additionally expects an *argument*, i.e., the name of the directory into which we want to change. Many different command line commands expect arguments, e.g. the command ***mkdir*** (short for *make directory*) can be used to create a new directory. It expects the *name of the new directory* as an *argument:*



```
Tims-MBP:course tim$ mkdir new_directory
Tims-MBP:course tim$ ls
total 16
drwxr-xr-x  2 tim  staff    68 17 Jul 10:32 Files
drwxr-xr-x  2 tim  staff    68 17 Jul 10:32 Workshops
drwxr-xr-x  2 tim  staff    68 17 Jul 10:32 meeting_notest
drwxr-xr-x  2 tim  staff    68 17 Jul 11:45 new_directory
-rw-r--r--  1 tim  staff   251 17 Jul 10:32 sequences.fasta
-rw-r--r--  1 tim  staff  1551 17 Jul 10:32 to-do.txt
```

As you can see in the output of *ls* there is now a new directory called *new_directory*. Similarly we can remove a directory with the command ***rmdir*** (short for *remove directory*) and give the name of the directory to remove as an argument:

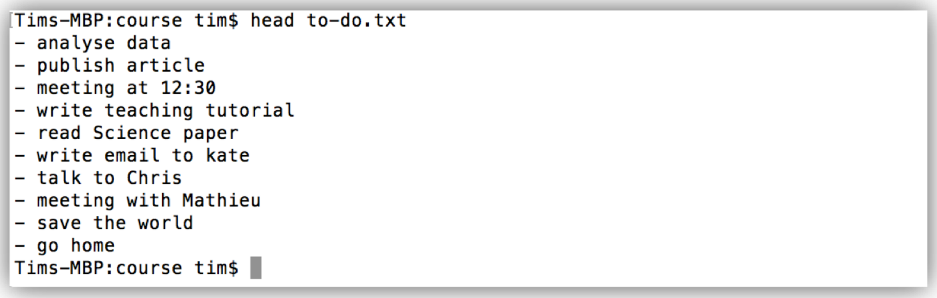


```
Tims-MBP:course tim$ rmdir new_directory
Tims-MBP:course tim$ ls
total 16
drwxr-xr-x  2 tim  staff    68 17 Jul 10:32 Files
drwxr-xr-x  2 tim  staff    68 17 Jul 10:32 Workshops
drwxr-xr-x  2 tim  staff    68 17 Jul 10:32 meeting_notest
-rw-r--r--  1 tim  staff   251 17 Jul 10:32 sequences.fasta
-rw-r--r--  1 tim  staff  1551 17 Jul 10:32 to-do.txt
```

Besides arguments the behavior of many commands can be altered by calling them with specific options. Let's have a look at the command ***head***. This command can be used to show the first lines of a text file. It expects the path to the file of interest as the argument and if no options are specified, it prints the first 10 lines of the specified file to the command line.

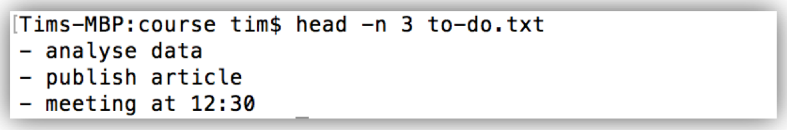
Command line - Commands

Let’s have a look at the first 10 lines of the file *to-do.txt* in our current working directory using *head*:



```
Tims-MBP:course tim$ head to-do.txt
- analyse data
- publish article
- meeting at 12:30
- write teaching tutorial
- read Science paper
- write email to kate
- talk to Chris
- meeting with Mathieu
- save the world
- go home
Tims-MBP:course tim$
```

As you can see I still have quite a few things to do before I can go home. However, if we want to list a specified number of lines using *head* we can do so by using the *–n* option of *head* and pass the number of lines we want to be listed as *a parameter*:



```
Tims-MBP:course tim$ head -n 3 to-do.txt
- analyse data
- publish article
- meeting at 12:30
```

Thus, a command line command consists of

- the command
- options (if applicable)
- option parameters (if appplicable)
- arguments (if applicable)

Note

In contrast to arguments options are passed to a command using a dash before the option, hence the *–n* in the above example. Additionally, some options expect a value/parameter, e.g. the number of lines to be printed in the above example. However, this is only true for some options.

Self Check 2

1. **How do you list the content of your current directory?**

2. **Which column of the *ls* output shows the size of a file ?**

3. **How do you create a directory ?**

4. **How do you add an option to a command?**

5. **How do you list the first 5 lines of a file ?**

Command line - Final

Congratulations, this is the end of the introduction!

If you read the previous pages you should now be able to use the command line to

- Know which directory you are in
- Navigate into other directories using absolute and relative paths
- Go one directory up
- List the content of a directory and interpret the output
- Make and remove directories
- List the first *n* lines of a file

Additionally, the last 2 pages of this booklet provide a *Cheat Sheet* of commonly used command line commands.

Ok, enough theory: it's time for some hands-on exercises to test your new knowledge.
If you haven't done this already please watch the tutorial videos on how to install *VirtualBox* on your computer, import the *envbio_course_vm.zip* and start the exercise on the next page of this PDF (installation instructions as well as the *envbio_course_vm.zip* can be found on the course page).

Note on *VirtualBox*
Depending on your computer you might get additional info or warning messages when opening the tutorial VirtualBox. Just accept those with *OK* or a similar button and you should be able to proceed wo the exercise.

Exercise

If you followed the installation instructions and started the *virtual machine* you should now see a command line window on your computer and you are ready to go. Please follow the instructions below. At the end of this exercise you will receive a password that you will need in the course!

To start the exercise type ***start_exercise*** on the command line. This will create a directory *env_course* in your home directory including a subdirectory called *experiment1* and several files.

To check whether you completed all tasks successfully type ***check_exercise*** and press return. If everything went well you will see a message telling you where you can find your password. Otherwise you will need to restart the exercise and try again.

To restart the exercise type ***restart_exercise*** and try again.

Tasks

1. **Create a directory called *new* in the *env_course* directory**
2. **List the content of directory *experiment1.* Copy the largest file in directory *experiment1* to the new directory you just created.**
3. **Rename the file *sample1.txt* in directory *experiment1* to *sample3.txt***

Hint

Use the *Cheat Sheet* at the end of this booklet to solve the tasks.

Your password is ____________________

Command line – Cheat Sheet 1

`cd` *`destination_path`*

Change directory.

Example 1: Change into directory */Users/tim/data*

```
cd /Users/tim/data
```

Example 2: Change into parent directory

```
cd ..
```

`mkdir` *`directory_path`*

Create a directory

Example 1: Create directory named *test* in the current working directry

```
mkdir test
```

`rmdir` *`directory_path`*

Remove a directory

Example: Remove the directory /Users/tim/data

```
rmdir /Users/tim/data
```

`rm file_`*`path`*

Remove a file

Example Remove the file *to-do.txt* in directory */Users/tim/*

```
rm /Users/tim/to-do.txt
```

`head file_`*`path`*

List first lines of a given file

Example 1: List first 10 lines of file *sequence.fasta* in the current directory

```
head sequence.fasta
```

Example 2: List first 3 lines in file to-do.txt in directory */Users/tim/*

```
head /Users/tim/sequence.fasta
```

Command line – Cheat Sheet 2

`grep pattern file_path`

Search a given file for a specific pattern (word or character) and print all matching lines to the command line

Example Print all lines of file /Users/tim/to-do.txt that contain the word “to” to the command line

```
grep “to” /Users/tim/to-do.txt
```

`pwd`

Print current working directory, i.e., the current location

`wc file_path`

Count the characters, words and lines of a given file

Example 1: List all characters, words and lines of file sequence.fasta

```
wc sequence.fasta
```

Example 2: print only the number of lines of file *?users/tim/sequence.fasta*

```
wc -l /Users/tim/sequence.fasta
```

`mv source target`

Move or rename a given file or directory.

Example 1: Rename file *sequence.fasta* in directory */Users/tim* to file *sequence_2.fasta*

```
mv /Users/tim/sequence.fasta /Users/tim/sequence_2.fasta
```

Example 2: Move file *to-do.txt* from directory */Users/tim/* to directory */Users/tim/data*

```
mv /Users/tim/to-do.txt /Users/tim/data/
```

`cp source target`

Copy given source file or directory to the given target directory

Example 1: Copy file to-do.txt from directory /Users/tim/data into directory /Users/tim/test/

```
cp /Users/tim/to-do.txt /Users/tim/test/
```