CSC Computing environment user guide
Version 11.11. 2013

This guide provides basic information about using the computing environment of CSC. The guide is intended for new CSC users who don't have strong background on using linux environment. This guide focuses to the issues that are identical or very similar in all of the CSC servers like, data management and interactive linux usage. Machine specific issues like compiler environments and technical details are discussed in the server specific guides like Taito User Guide and Sisu User guide.
# Table of Contents

CSC Computing environment user guide.................................................................................. 1

1. Introduction.......................................................................................................................... 5
   1.1. Notation used in this guide............................................................................................. 5
   1.2 Getting access to CSCs' resources.................................................................................. 5
      1.2.1 Registration as a CSC user......................................................................................... 5
      1.2.2 Applying for computing and storage resources......................................................... 7
      1.2.2.1 Computing services................................................................................................. 7
      1.2.2.2 Ida storage service.................................................................................................. 7
      1.2.2.3 Database service...................................................................................................... 7
      1.2.2.4 Language Bank of Finland..................................................................................... 7
      1.2.2.5 User rights for FGI computing grid......................................................................... 7
      1.2.2.6 International collaboration..................................................................................... 8
   1.3 Connecting to the servers of CSC.................................................................................... 8
      1.3.1 Scientist's User Interface............................................................................................ 8
      1.3.2 Terminal connections................................................................................................. 9
      1.3.3 NoMachine Remote Desktop..................................................................................... 11
   1.4 Managing your CSC user account................................................................................... 12
      1.4.1 Passwords................................................................................................................ 12
      1.4.2 Groups and quotas.................................................................................................... 13
      1.4.3 Managing project information with MyProjects tool.............................................. 13
      1.4.4 Checking user groups and saldo with command line tools..................................... 14
   2. Linux basics for CSC......................................................................................................... 17
      2.1 Using Linux in command line....................................................................................... 17
         2.1.1 Files and directories in linux................................................................................... 17
         2.1.2 Structure of linux commands................................................................................. 17
         2.1.3 Basic commands for using directories.................................................................... 18
         2.1.4 Basic commands for files...................................................................................... 20
         2.1.5 Special characters................................................................................................... 22
         2.1.5.1 Wild card characters.............................................................................................. 22
         2.1.5.2 Redirecting standard input and output................................................................. 23
      2.2 Editing text and viewing images................................................................................... 23
         2.2.1 Nano....................................................................................................................... 23
         2.2.2 Emacs..................................................................................................................... 24
         2.2.3 Image and PostScript viewers................................................................................ 25
      2.3 Working with bash shell................................................................................................ 26
         2.3.2 Automatic Tab completion...................................................................................... 27
         2.3.3 Stopping programs and running programs in background..................................... 28
      2.4. More commands for managing files............................................................................ 29
         2.4.1 Using find to locate files........................................................................................ 29
         2.4.2 File command tells the file type.............................................................................. 29
         2.4.3 Count rows and characters with wc....................................................................... 30
         2.4.4 Comparing two files with diff............................................................................... 30
         2.4.5 Using checksums to verify successful data storage or transfer................................ 30
         2.4.6 Encrypting files with gpg....................................................................................... 31
         2.4.7 Managing access permissions of files and directories........................................... 32
         2.4.8 Managing access permissions with Scientist's User Interface.................................. 33
         2.4.9 Managing access permissions in command line usage........................................ 34
      2.5 Commands for data processing..................................................................................... 35
4. Running applications at the servers of CSC

2.5.1 Grep selects rows that match given string..........................................................35
2.5.2 Using sed to select rows by row number..........................................................37
2.5.3 Simple column selection with cut.................................................................38
2.5.4 Using awk to work with columns.................................................................38
2.5.5 Using sort to order rows..............................................................................40
2.5.6 Removing duplicate rows with uniq.............................................................41
2.5.7 Replacing characters with tr.......................................................................42
2.5.8 Replacing words and strings with sed.........................................................44

2.6 Packing and compression tools........................................................................45
2.6.1 Tar and gtar: packing several files into one file...........................................45
2.6.2 Compressing files.........................................................................................48
2.6.2.1 Gzip and gunzip.....................................................................................48
gzip example........................................................................................................48
2.6.2.2 bzip2 and bunzip2................................................................................49
2.6.2.3 Zip and unzip: combined compression and file packing tool..................50
Zip example............................................................................................................51

2.7 Linux bash scripts..........................................................................................52
2.7.2 Variables and arrays.....................................................................................53
2.7.3 Quotation marks............................................................................................56
2.7.4 Loops and conditional statements...............................................................57
2.7.5 Printing the output.........................................................................................60

3 Directories and data storage at CSC..................................................................62
3.1 User specific directories at the servers of CSC.................................................63
3.1.1 Monitoring disk usage.................................................................................63
3.1.2 Home directory, $HOME..............................................................................64
3.1.3 Work directory, $WRKDIR.........................................................................65
3.1.4 Temporary directory, $TMPDIR....................................................................65
3.1.5 Software installation directory $USERAPPL...............................................65
Example 1. Installing your own version of program MCL program......................66
Example 2. Installing and using your own Perl module to Taito............................68
Example 3. Installing and using your own Python module at Taito........................69

3.2. Archiving data to the HPC archive and IDA storage services..........................69
3.2.1 Using HPC archive.......................................................................................70
Example 1. Storing data to the HPC archive server..............................................71
Example 2. Retrieving data from the archive server..............................................73
3.2.2 Configuring the connection to IDA..................................................................73
Example 3. Storing data to the IDA server.............................................................74

4. Running applications at the servers of CSC.....................................................76
4.1 Application software selection at CSC..............................................................76
4.2 Environment module systems.........................................................................77
4.3 Batch jobs.........................................................................................................78
4.3.1 Constructing SLURM batch job files............................................................79
4.3.2 Using SLURM commands to execute batch jobs.........................................82

5. Moving data between CSC and local environment...........................................84
5.1 Data transport with Scientist's User Interface..................................................84
5.1.1 Managing files with My Files tool...............................................................84
5.1.2 Transporting large files using the SSH Console tool....................................87
5.2 Copying files from Linux and Mac OSX machines with scp.........................89
5.3 File transfer in Windows with WinSCP.........................................................90
5.3.1 Getting started...............................................................................................90
5.3.2 Uploading files to CSC environment.................................................................91
5.3.3 Downloading files from CSC environment.....................................................92
5.3.4 Further documentation..................................................................................93
5.4 Using rsync for data transfer and synchronization............................................93
   Rsync example 1. copying a directory structure from local computer to CSC.........94
   Rsync example 2. Synchronizing directories within a one server.........................95
   Rsync example 3. Copying a large file from remote site to CSC..........................95
5.5 Using wget to download data from web sites to CSC..........................................96
5.6 Using Funet FileSender to share and transport files..........................................97
   5.6.1 Uploading data to FileSender.................................................................97
   5.6.2 Download data from FileSender.............................................................98
5.7 Remote disk mounts.........................................................................................99
Appendixes.............................................................................................................101
   1 List of linux commands discussed in this guide.............................................101
1. Introduction

1.1. Notation used in this guide

In this guide, the **boldface** font is used indicate a command or parameter, that is appearing for the first time in the normal text of a chapter. For example: "Use command `ls` to see the content of a directory". After the first appearance commands, parameters and file names are typed with *italic* font. Italic font is also used to introduce new terms within the text.

To emphasize how to write a command the courier font is used. For example:

Use following command to log in to the taito.csc.fi cluster:

```
ssh taito.csc.fi.
```

Generic arguments given to commands are indicated with an italic font. For example:

```
rm file
```

In the command examples the output is also displayed within a grey box. For example

Command

```
ls
```
gives following results

```
file1
file2
```

Some examples use a notation where both command and the results are in the same box. In these cases the command prompt is shown and the actual command is displayed with **bold face** letters. For example:

```
kkayttaj@taito:~> ls
file1
file2
```

Note that here the command prompt ( e.g. `kkayttaj@taito:~> ` ) is not part of the command.

1.2 Getting access to CSCs' resources

1.2.1 Registration as a CSC user

The services that CSC offers have been listed in our [Customer web pages](#). In most cases the usage of the computer resources of CSC is free for academic users but you need register as a CSC user. For registration you can use your university account (Haka-account) to log in to the Scientist's User
Interface to fill the registration form:

- [https://sui.csc.fi/signup](https://sui.csc.fi/signup)

In the registration you need to agree to follow the *General Terms and Conditions Concerning the Use of CSC’s services*. After successful registration you will get a CSC username and password. Generating a new CSC username typically takes about one day. Once you have the CSC username and password, you can start using the following services:

- [Scientist’s User Interface](http://sui.csc.fi), i.e. CSC’s extranet service: http://sui.csc.fi
- Application server hippu.csc.fi (through ssh connection)
- Chipster - life science oriented data analysis environment: [http://chipster.csc.fi](http://chipster.csc.fi)

As a registered CSC user, you can access a wide selection of software and databases available at CSC. However, your user account has a limited quota for computing (1000 core hours on Hippu). For heavier computing tasks, you should fill in the *Application form for computing services* (below).

If you don't have a [Haka](https://haka.csc.fi) account, you should use *Application form for computing services* for getting access to CSC.

---

**Figure 1.1** registration form for CSC users in Scientist’s User Interface.
1.2.2 Applying for computing and storage resources

1.2.2.1 Computing services

Researchers that need to use the clusters and supercomputers of CSC should fill the Application form for computing services. (This application form is not available as electronic version)


This form is used to open a computing project to the servers of CSC. One project typically includes 1-10 user accounts. The project has a common computing quota that can be extended by application. You can apply user accounts for several computing servers which your project members need access to:

- massive parallel supercomputer (sisu.csc.fi)
- Supercluster (taito.csc.fi)
- Application server (hippu.csc.fi)

Use of these servers will consume the computing quota granted to the project. Most of the application software and databases are installed on these servers. You may have to agree to follow service specific terms and conditions for your use of application software and databases. The user accounts also includes access to Scientist's User Interface and HPC archive. New members can be later on added to an existing computing project.

Projects that require exceptionally large computing or storage resources can apply extra resources through periodically announced grand challenge and DECI-prace calls.

1.2.2.2 Ida storage service

The Ida service is intended for long term storage of scientific data. Once you have registered as a CSC customer, you can apply for storage space in Ida with following application form:

- http://www.csc.fi/sivut/ida/esittely/ida_kayttolupahakemus_20130123

1.2.2.3 Database service

CSC's database server, kaivos.csc.fi, provides researchers a possibility to create their own MySQL databases. To apply for a new database you need to have a computing project at CSC as the database service always relates to customer's computing project.


1.2.2.4 Language Bank of Finland

Language Bank of Finland gives access to linguistic research material and software. For further information see the linguistics home page.


1.2.2.5 User rights for FGI computing grid

Grid usage is possible by first obtaining a grid user certificate and then joining fgi.csc.fi or other Virtual Organization. Obtaining a certificate for grid computing does not require registration as a
CSC user. More detailed instruction can be found from FGI User Guide and FGI home pages: http://fgi.csc.fi

### 1.2.2.6 International collaboration

A close collaborator of a Finnish research group, who works abroad, can apply for CSC user account for a fixed period. However the project manager (principal investigator) should be working in Finland and he has to have the right to use CSC's resources (e.g., a professor in a Finnish university). More detailed instruction about this can be found from the CSC web pages.

CSC can also facilitate international research visits through the HPC-Europa2 project (Pan-European Research Infrastructure on High Performance Computing) which offers the opportunity for Finnish universities to host European visiting researchers while CSC makes its supercomputers and other support available to visitors.

### 1.3 Connecting to the servers of CSC

You can connect the servers of CSC either by using the web based Scientist's User Interface extranet service or through Linux terminal connections. While the Scientist's User Interface provides an easy and intuitive way to connect to CSC, it allows you to use only a very limited set tools for working with your data. Terminal connections and Linux command line usage still provides the most versatile and efficient tools to manage and analyze data in the servers of CSC.

#### 1.3.1 Scientist's User Interface

Scientist's User Interface is an extranet service that allows the customers of CSC to access the CSC computing environment through a normal web browser. The URL of the Scientist's User Interface (SUI) is https://sui.csc.fi and it can be used for following tasks

- monitoring and managing your user account at CSC
- managing files in the servers of CSC
- managing files in the IDA storage service
- Launch SSH console sessions
- Use the Soma2 drug discovery environment
- Use Lemmie language bank corpora interface and DMA -Digital Morphology Archive
- Use PalTuli Geospatial Data Service
- Download software and data sets provided by CSC
- Create batch job scripts for the computing servers of CSC.
- Manage and store grid certificates

You can use either your HAKA account (local university account) or CSC user account to log into Scientist's User Interface. The My Files file manager is discussed more in detail in chapters 2.4.8 and 5.1 of this guide.
1.3.2 Terminal connections

A terminal connection to CSC should be opened using a secure terminal program that encrypts the data traffic between the user and CSC. In Linux and Mac OSX systems you can use ssh program to connect to CSC. In Windows machines you can use for example the PuTTY open source program to open the connection to CSC. If you don’t want install a local ssh terminal program, you can use the SSH console tool in Scientist’s User Interface (more details in chapter 5.1.2).

For example to log into Hippu, open a new terminal screen and give command:

```bash
ssh hippu.csc.fi -l csc_username
```

If you wish to use also X11 based graphical tools give command:
ssh hippu.csc.fi -X -l csc_username

or in some cases, if the command above does not work, try:
ssh hippu.csc.fi -Y -l csc_username

When you stop working at the servers of CSC, you should first close the connections to CSC with command:
exit

In addition to the plain text based terminal connections you may need be able to use also X-term connections. X-term connection can be used to enable viewing image files or launching a graphical interface on the servers of CSC. To be able to use X-term based tools your local computer must have an X11 server (often called as X window server program) installed and running. In linux machines X11 is normally installed and running automatically. In Mac OSX machines the X11 software is not installed by default, but it can be installed from the Mac OSX installation disks. In Windows machines, an X server program must be installed to the users local machine. In addition to several commercial xterm programs there are also some free ones like Xming.

Once your X server program is running, open a terminal connection to CSC (remember to use the -X or -Y option or Tunnel X11 setting), and test by running command:
xterm &

This should open a new xterm connection to your screen.

![Linux terminal window](image)

**Figure 1.3** Linux terminal window
1.3.3 NoMachine Remote Desktop

NoMachine Remote Desktop provides an alternative way to use remotely graphical interfaces at the servers of CSC. The service works with a NoMachine client which is installed on the researcher's computer. No X-term software is needed in the local computer. The NoMachine client first connects to a NoMachine server running at nxlogin.csc.fi or at nxkajaani.csc.fi. You can use both NoMachine servers to open a connection to the servers of CSC. The nxkajaani.csc.fi server locates in the CSC Datacenter in Kajaani and it is recommended server to access Sisu or Taito (that locate on Kajaani). Similarly, nxlogin.csc.fi locates in Espoo and is recommended for Hippu and Vuori users.

After the login, a remote desktop opens to your screen. From this remote desktop you need to further connect to the actual computing servers (by right clicking the desktop) and then start the applications with normal terminal commands.

We recommend NoMachine in for remote usage of graphical interfaces in stead of X11 connections. It often speeds up the graphical response 10-100 fold with respect to the normal X11 connections. Further FreeNX client is often easier to install to a Windows machine, than an X-emulator. Instructions how to set up and use the FreeNX remote connection can be downloaded form the Scientist's User Interface:

- FreeNX remote desktop instructions

![FreeNX remote desktop instructions](image)

**Figure 1.4** Matlab opened in the FreeNX remote desktop.
1.4 Managing your CSC user account

1.4.1 Passwords

To be able to connect the CSC servers you need a CSC user account and password. These you can get by filling the applications for computing and application software usage. Once your account is opened the password of your account is set to by normal mail. One of the first things a new CSC user should do is to change the password given by CSC in to a new one. You can change your CSC password either with Scientist's Interface or in Linux command line. A good password should meet following requirements:

- contain eight or more characters
- should not contain any meaningful word or year number
- should contain a mixture of lower-case and upper-case letters and numbers and special characters
- should not be based on keyboard geometry
- should not be used elsewhere

In the Scientist's User Interface, the password can be changed using the Scientist's User Interface | My Account tool. You can check you personal information by clicking your name in the upper right corner of the Scientist's User Interface. This opens My Account page, that you can use to check and modify information related to your account. In the list on the right side of the page you can find a link to CSC password tool that you can use to change your password.

![Manage My Account](Manage_My_Account.png)

**Figure 1.2:** Password set up in My Account tool in Scientist's User Interface

In the linux command line usage, the password is changed with command:
passwd

When you update your password, the passwd Command. This command asks first the old password and then two times the new password. When the passwd command is executed, the new password will be taken in use in all of the servers of CSC and in the Scientist's Interface (your Haka password is not changed). After running the command it may take up to one hour until the password information has been updated in all the servers of CSC.

1.4.2 Groups and quotas

Each computing project at CSC has a limited computing quota. At CSC one research project normally forms one linux user group. However one user account may belong to several projects and linux groups. If your account belongs to more than one project, you can direct your CPU time consumption to the desired project by changing your primary group. The primary group is server specific: your account can utilize quotas from different projects in different servers and can change the primary group when needed.

Once the granted quota is used the manager of the project should apply for more computing time. The computing quota is measured with billing units.

- In Hippu and Vuori a one-hour job using a single processor core consumes one billing unit (bu), i.e. 1 CPUh = 1 bu.
- In Sisu and Taito a one-hour job using a single processor core consumes two billing units (bu), i.e. 1 CPUh = 2 bu.

Example: A batch job on the sisu.csc.fi supercomputer with 64 processor cores, lasting for two hours, consumes 256bu. Computing resources are shared by the project. The members of the project use resources in common.

In the CSC environment, the quotas of customer projects are updated four times a day: at 8:15, 12:15, 16:15 and 00:15. Further, the computing quota is contains data from completed batch jobs. The jobs in queue or running are not yet included in the quota information.

When the project's compute resources drop below 20% of the original quota, you will receive an automatic warning of the format:

ATTENTION: You have less than 20% CPU time quota left on your default account. Please apply for more CPU time quota.

It is advisable to apply for more CPU time for your project well before the current compute resources are used up. However, those jobs that are already running are completed, even though the project quota becomes negative during the run.

1.4.3 Managing project information with MyProjects tool

The My Projects tool in Scientist's User Interface provides information about the current status of the project or projects that the user has at CSC. The projects that the user has at CSC are listed at the top of the My Projects interface. The CPU quota of the project is shown in the CPU quota display box in lower right corner of the interface. This display box also contains link to a tool that can be used to apply for more CPU time.

The CPU Usage display box allows you to study the CPU usage of the project in many different ways. You can for example define a time period to be studied and check how much each of your project members has used the quota. You can also apply for more computing time with the My Projects tool.
If your account belongs to more than one project, you may direct your CPU time consumption to the desired project by changing your primary group. To change your primary group in some server, first select a project from the My Projects list by double clicking the project name. Then open a host list by double clicking the Hosts: Edit link in the Project information display box. This opens a server list from which you can select the servers where the currently selected project will be used as the primary project (i.e. the billing project).

### 1.4.4 Checking user groups and saldo with command line tools

The Unix command `groups` shows the groups your user account belong to. First comes the primary group that indicates the project quota that your compute jobs are currently consuming. The
command **id user_account** shows which project numbers corresponds to the groups.

If your account belongs to more than one project, you may direct your CPU time consumption to the desired project by changing your primary group. The primary group can be different in different servers. To change the primary group in Hippu and Vuori:

1. Give the Unix command `chprigrp` and give the password to the server.
2. Check your current primary group: *Your current group is:*
3. Type the new primary group upon request: **New primary group:**
   (If you don't want to change your primary group, press Enter.)
4. Log out from the server and log in again to activate the change.
5. Note that the primary group change will become active within an hour, and is server specific.
6. Change the primary group again if you start to work for a new project.

Example: Changing the primary group on Hippu

```
testuser@hippu1:~> chprigrp
[connecting to the password server]
Password:
Changing primary group for test-user on host hippu.
Press return to keep your current primary group.
Your current group is: project1
You belong to groups: project1, project2
New primary group: project2
[primary group changed]
[Changes become active within an hour]
testuser@hippu1:~> logout
```

The `chprigrp` command is not yet in use in Sisu and Taito. Please use the MyProjects tool described above, if you need to change the primary group in those servers.

At the computing servers of CSC, you can use `saldo` command for monitoring your project's usage of computing resources. Saldo prints user's projects' total usage of CSC resources, and project members' monthly usage on each different servers in both billing units (bu) and CPU seconds.

Below is shown the output of saldo, which shows the usage of a sample project on 5.12.2012 at 16:15. The project's code is `csc1000`, and name *Uudet mallit rautalangasta*. The project manager is Veikko Vetäjä, and the users are vvetaja, ppostdoc and atutki (if the user has not used any resources, his/her usage is not printed). The project has started on 6.4.2009, and will end on 2.9.2013. The last grant of computing resources was made on 4.1.2011.

```
vvetaja@hippu1 ~> saldo

----------------------------------------------
Saldo for year 2012 month 12
----------------------------------------------
Project csc1000 Uudet mallit rautalangasta Veikko Vetäjä
start 06.04.2009 end: 02.09.2013 budget: 4.1.2011
CSC budget: 46291 used: 42155.78 remain: 4135.22
  Cpu usage Cpu secs Bu
----------------------------------------------
```
Contents of the saldo print:

- **CSC budget**: Cumulative project quota from the beginning of the year (46291 billing units, bu)
- **used**: Cumulative project usage from the beginning of the year (42155.78 bu)
- **remain**: Remaining computing resources of the project at the time saldo was printed (in billing units)
- **Cpu usage** (monthly listing per user and per server):
  - The monthly usage of all active members of the project: project, userid, his/her usage in CPU seconds (CPU secs) and billing units (Bu), and the used server
  - At the bottom is shown the project’s monthly total usage in CPU seconds and billing units.

In this case the saldo report shows that Veikko Vetäjä should apply for more CPU time for his project, because the current compute resources will be used up soon.

Saldo has four options:

- **-p projid** prints a saldo report for the selected project. You need this option if you belong to several projects.
- **-m month** can have values between 1–12. You can print the usage of the selected month up to one year backwards. E.g. `saldo -p csc1000 -m 1` prints the project usage from January.
- **-u userid**: With this option, user’s monthly report is printed.
- **-h file-name** generates an HTML file of the report.
2. Linux basics for CSC

The servers of CSC use Linux operating systems. While the users local computer is normally used mainly through a graphical user interface, the remote usage of the servers of CSC is in most cases based on working on command line: instead of using mouse and command menus, the user types commands to a screen called terminal or console. Even though more and more tools provided by CSC can be used through graphical interfaces, using the basic linux commands is in many cases still the most efficient way.

This chapter provides introduction to the usage of the Linux environment of CSC. The application server, hippu.csc.fi, is used in the examples but the same commands can be used in the other servers of CSC too. If you are using Linux or Mac OSX machine, you can use most of the commands in also your local machine. Once you are familiar with the basic linux commands, you can continue to chapters that go deeper to the file and data management commands of linux systems.

The default command shell for new users in the CSC computing environment is bash. Before 1st June 2012 the default was tcsh. Users can change their shell using the chsh command. CSC fully supports both bash and tcsh shells. Customers using other shells are also supported, but may be asked to reproduce any issues in bash or tcsh. This guide is accurate for bash shell only. If you are using some other linux command shell like tcsh, csh, sh or ksh, all the details may not apply.

2.1 Using Linux in command line

2.1.1 Files and directories in linux

The file systems in linux machines is based on hierarchical directory tree. There is one root directory in to which you can refer with slash sign (/). All the files and directories locate in the subdirectories of this directory so that each file has a unique combination of name and directory path. Also the commands that user gives are executed in some directory. The directory where a user locates is called as current working directory.

Normally you do not need to know the explicit directory paths when you work in the CSC environment. It is enough to know the locations of the files in users own disk areas. The user and project specific disk areas are represented in chapter 3. However, you should remember that many disk areas in CSC environment can be accessed from several different servers (e.g. the users home directory) while some areas are server specific. In the case of shared disk areas the path to a certain file may be different in different servers.

2.1.2 Structure of linux commands

Once the terminal connection to CSC, e.g. to Hippu-server, has been opened, the remote server is used with linux commands. The standard structure of a linux command is:

```
command -options argument1 argument2 ...
```

The command is executed by pressing the return key (Enter). The names and functions of options and arguments depend on the linux command. In many cases you can run the command without any options and arguments. Options are used to modify the actions that the command performs. Arguments are used to define the files, directories and values that are used as input parameters and to define where the output is written.
For example command `ls` can be used with several options and arguments. Running plane command `ls` lists the content of a directory in alphabetical order. You can modify the output of the command for example by using option `-t`. With this option the directory content list is ordered by the age of the file (time stamp). If no argument is given, `ls` prints out the content of the current working directory. By giving an argument to the `ls` command, user can define what directory should be studied. For example command:

```
ls -t /wrk/kkayttaj
```

will list the content of directory `/wrk/kkayttaj`. In case the command, argument or options contain errors, the command will not be executed when the return key is pressed. In stead an error message is printed to the screen. Thus having errors in the command normally does not cause any major problems. The output of commands depends on the command but in many cases, no output means that command was successfully executed.

Most of the linux commands have their own manual page that can be studied with `man` command. For example the manual page of `ls` command could be studied with command:

```
man ls
```

The manual pages can be very detailed and technical. However, often you do not need to read and understand all the details on the manual pages, but in stead you can just see what command line options there are available for the command and then start testing them in practice.

Often the most difficult thing in using linux is knowing the name of suitable command. Below we provide introduction to the most frequently used linux commands. You can also use command `apropos` to find suitable command. Apropos lists those linux commands whose short description lines match the text that is given as a command argument. For example to look for commands that are processing `pdf` files you could give command:

```
apropos pdf
```

Note that the listing that apropos prints out includes only linux commands but not application program names. Thus, the sample command above would produce a list that contain many pdf conversion commands but not the pdf viewing programs like `acroread` or `evince`.

### 2.1.3 Basic commands for using directories

Table 2.1 lists the commands that are most frequently used for moving in the directory hierarchy and managing it. Below are some examples of directory related commands.

**Table 2.1: Basic directory commands**

<table>
<thead>
<tr>
<th>Name</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cd</code></td>
<td>directory</td>
<td>Change current working directory</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>directory</td>
<td>Lists the content of a directory</td>
</tr>
<tr>
<td><code>pwd</code></td>
<td></td>
<td>Print the directory path of current working directory</td>
</tr>
<tr>
<td><code>mkdir</code></td>
<td>directory</td>
<td>Create a new directory</td>
</tr>
<tr>
<td><code>rmdir</code></td>
<td>directory</td>
<td>Remove an empty directory</td>
</tr>
</tbody>
</table>

When you log in to a server at CSC, you will first locate in your home directory. You can check your location i.e. the path of the current working directory with command `pwd` (abbreviation from Print current Working Directory). However, you do not have to remember the location of your home.
directory (see cd command).

The content of the directory is listed on your screen with command ls. Plain ls command just lists the names of the files and directories in your current directory. You can get more information about the files and directories with command ls -la. The -l option produces a long directory listing that in addition to the name contains also information of access settings, size and the modification dates of the files and directories. The option -a defines that all files, including also the settings files that start with dot (.) character, are listed. Below is a sample output for ls -la command:

```
kkayttaj@hippu1:~> ls -la
```
```
total 26914
  drwx------  3 kkayttaj csc  10 Dec 22 09:12 .
drwxr-xr-x  20 root root   0 Dec 22 09:12 ..
drwx------ 42 kkayttaj csc   472 Dec 22 09:07 ..
-rwxr-x---+  1 kkayttaj csc 1648 Dec 22 09:01 .cshrc
-rw--------+  1 kkayttaj csc  93 Dec 22 09:01 .my.cnf
-rw--------+  1 kkayttaj csc  48 Dec 22 09:05 Test.txt
-rw--------+  1 kkayttaj csc 878849 Jan 19 2009 input.table
  drwxr-xr-x+  2 kkayttaj csc  2 Dec 22 09:11 project1
-rw--------+  1 kkayttaj csc 26432051 Dec 22 09:08 results.out
-rw--------+  1 kkayttaj csc  25 Mar 27 2009 sample.data
-rw--------+  1 kkayttaj csc  49 Mar 27 2009 test.txt
```

The first output row: total 26914 tells that the total size of the files in the directory is 26914 KB. In the list, the first character tells if the item is a file (-) or directory (d). The next nine characters display the access permissions of the files (see the chmod command chapter for more details). The next columns show the number of links pointing to the item, owner, user group, size in bytes, modification time and finally the name of the file or directory.

By default the files are presented in alphabetical order. You can order the results by the modification time with option -t or by size with option -S (note: upper case S, not lower case s). Two other frequently used options are -h (Human readable) which prints out the sizes of larger files in megabytes or gigabytes and -r which means reverse sorting order. For example command:

```
ls -ltrh
```

is very handy when you want to check what files have recently been modified or created. ls and pwd commands do not modify your files in any way so you can use them always when you want to know where you are and what files your current directory include.

Command cd directory_name moves you from directory to another. For example the user kkayttaj could go to his wrk directory with command:

```
cd /wrk/kkayttaj
```
or

```
cd $WRKDIR
```

In the latter command automatically defined environment variable $WRKDIR, that contains the explicit directory path, is used to define the target directory.

New directories can be created with command mkdir directory_name. For example command:
mkdir project1

Creates a new directory called *project1*. You can use *ls* command to check that the directory was created. Now you can go to this directory with command

cd project1

You can come back from the project1 directory with command:

cd ..

Note the space between *cd* and the dots in the command. One dot (.) and two dots (..) have special meaning in the Linux commands. One dot (.) means the *current directory* and (..) means the directory that is one step lower in the directory tree i.e. the directory where current directory locates. Executing the *cd* command without any arguments will always move you back to your home directory, regardless where you are in the directory tree. An empty directory can be removed with command *rmdir* directory_name. For example:

```
rmmdir project1
```

### 2.1.4 Basic commands for files

In a very basic level a file in a Linux system is just a string of bytes, were a byte consists of eight bits. So called text files the contain only bytes that can be interpreted as text characters using ASCII coding. Thus these files can be considered to consist of lines of text. In so called binary files also non ASCII bytes are used and the text can't and is not intended to be converted to text. Typical examples of binary files are compiled programs, images or compressed files. Normally users work mostly with text files and also in the examples of this guide we normally assume that the files contain some kind of text data: letters or numbers.

Each file has a name. Name can in principle be any combination of characters. However, several characters have special meaning in Linux commands (e.g. ?, *, #, see paragraph 2.1.5 below) and thus using these characters in file names may cause problems. We recommend that you use only normal letters (lower or upper case), numbers, dot (.), dash (-) or under score (_) characters in file and directory names at CSC. Also the space characters in file names cause often problems. We recommend that the space characters are replaced with underscores, for example: *new_file.txt*. Note that Linux is case sensitive: lower and upper case characters are not considered equal and for example names *New_File.txt* and *new_file.txt* refer to different files.

In Linux, usage of name extensions like .doc or .txt is not obligatory. Most of the Linux tools do not require specific extensions to be used. However, on the long run, using systematic naming conventions, including illustrative name extensions, makes the file management easier.

**Table 2.2:** Commands for using files

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>file_names</td>
<td>Print the content of the specified file of files to the standard output (your screen)</td>
</tr>
<tr>
<td>chmod</td>
<td>file_names</td>
<td>Change the access permissions of a file</td>
</tr>
<tr>
<td>cp</td>
<td>file_name1, file_name2</td>
<td>Copy the file content to a new file or new location</td>
</tr>
</tbody>
</table>
You can study the content of text files with commands **cat**, **more** and **less**. These commands are safe to use as they don't modify the files in any way. For example you could study the `.bashrc` file in your home directory with commands:

```
cat .bashrc
more .bashrc
less .bashrc
```

The **less** pager program is more advanced than **more**. Less can browse the text also backwards either one row at a time by pressing `k` or one screenfull at a time by pressing `b`. You can also search a text string from the document by using slash (`/`) character. For example, to locate a string `ABC` from the file in **less** program type `/ABC` and the press `Return`.

Commands **head** and **tail** can be used to see just the first or last rows of a file. By default these commands print out 10 lines, but you can change this by giving the number of rows to be printed as an option to a **head** or **tail** command. For example, to check the 30 last rows of a file called `run1.log` give command

```
tail -30 run1.log
```

Copying files to a new file or to another directory is done with command **cp** (copy). Below is two examples of copy commands:

```
cp output.dat output_copy.dat
cp output1.dat output2.dat results/
```

The first command makes a copy of file `output.dat` to a new file called `output_copy.dat`. In the second example the two files: `output1.dat` and `output2.dat` are copied to existing directory called `results`. Command **mv** (move) is used to rename or move the files to another location. For example:

```
mv output.dat output_copy.dat
mv output1.dat output2.dat results/
```
would create the same new files as `cp` sample commands. However in the case of `mv` the original files `output.dat`, `output1.dat` and `output2.dat`, would be removed from the current working directory.

Files are removed with command `rm file-name`. For example

```
rm output_copy.dat
```

In the CSC environment, the `rm` command asks the user to confirm that he/she really wants to remove the file:

```
kayttaj@hippu1:~>rm output_copy.dat
rm: remove output_copy.dat (yes/no)?
```

You can answer `y` (yes) or `n` (no). Note that this confirmation step is not necessary in use in your local linux environment. You can skip the confirmation query with option `-f`. However you should use this option with caution as the `rm` command will remove the file immediately permanently.

### 2.1.5 Special characters

Some characters have special functions in linux commands. In the following paragraphs we present characters that are used for redirecting standard input and output or used as so called wild card characters.

The `$` sign, that is used as an indicator of a variable name, `#` mark that is used to place comments and different kind of quotation marks are discussed later on in the [linux scripting chapter](#).

#### Table 2.3: Commonly used special characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>Indicates the beginning of a shell variable name</td>
</tr>
<tr>
<td></td>
<td>Pipes standard output to the standard input of the next command</td>
</tr>
<tr>
<td>#</td>
<td>Starts a comment</td>
</tr>
<tr>
<td>&amp;</td>
<td>Executes a process in the background</td>
</tr>
<tr>
<td>?</td>
<td>Matches one character</td>
</tr>
<tr>
<td>*</td>
<td>Matches any string (including an empty string)</td>
</tr>
<tr>
<td>&gt;</td>
<td>Output redirection operator</td>
</tr>
<tr>
<td>&lt;</td>
<td>Input redirection operator</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Output redirection operator (to append to a file)</td>
</tr>
<tr>
<td>\</td>
<td>ignore the possible special function of the following character</td>
</tr>
</tbody>
</table>

#### 2.1.5.1 Wild card characters

In linux commands, question mark (`?`) and asterisk (`*`) are used as so called wild card characters. They can be used to define arguments that match many files or directories. When used in the command arguments the `?` sign is interpreted as *any single character* and `*` sign as *any string of characters*. For example command:

```
ls test?.input
```

Would produce a list of files that has a name: `testany character.input`. Thus files with names `testA.input` and `test4.input` would be listed, but file names like `test10.input` or `testOld.input` would be ignored. In stead, command:
ls test*.input

would list all of the files mentioned above as * matches any string. Now the only limitations would be that the command must start with string test and end with string .input.

2.1.5.2 Redirecting standard input and output

Characters less than (<), greater than (>), >> and pipe (|) are used to control the standard input and output. The less than (<) instructs the command to read data for standard input from a file defined after that < character instead of the keyboard.

The greater than character (>) would direct the standard output of to a new file in stead of the display. For example command:

ls test*.input > input_files

Would produce a new file called input_files that would contain the names of files that start with string test and end with .input. Using two greater than signs, with no space between them, ( >> ) would append the results of a command to the end of an existing file or, if the file does not yet exist, direct the output to a new file.

The pipe character (|) directs the standard output of the command to the standard input of the next command. With this function you can combine several linux commands into a command chain. For example if your file listing does not fit to one screen, you could redirect it to less command so that you can study and browse it one screenfull at a time. This kind of redirecting could be done with command

ls -l | less

As another example, we could use grep command as a post processor for ls command and pick for example those files that have been created on August 14th.

ls -l | grep "Aug 14"

2.2 Editing text and viewing images

Editing text files by typing in linux systems can be done with so called text editor programs. Editors are normally used to modify or create relatively small files like programs, parameter files or small input files. Systematic modifications to large input or output data files are normally done with other tools.

Several text editors are available in the CSC computing environment. Here we briefly introduce just two of them: nano and emacs. Other frequently used text editors include vi, vim and gedit. In addition to these general purpose text editors, many application programs include their own special editing tools.

2.2.1 Nano

Nano is a simple and handy tool for editing small files. It is easy to use and it suits well as the first text editor of a novice linux user. Nano is a text based editor: it does not include any mouse based functions or command menus. Nano starts with command:

nano text_file_name

This command opens the given file to the editor or creates a new empty file if the file does not
already exist.  

When the nano editor has started, you can see the basic commands listed in the bottom of the screen. The listed commands are executed by pressing simultaneously the control key: Ctrl, into which the program refers with ^ sign, and some of the listed keys. For example you can read the exit from the program with Ctrl-x, save the changes with Ctrl-o or read a new file to the editor with Ctrl-r. If you already have some text in nano, Ctrl-r will add the content of the selected file to the end of the previously created text. With Ctrl-v you can browse the text one page forward and with Ctrl-y one page backwards. Ctrl-W command can be used to search certain strings (words) from the text. Ctrl-k cuts one line of text. To cut larger regions, first mark the beginning of the text to be copied with Ctrl-6 then move the cursor to the end of the area and press Ctrl-k. To paste the text that was cut, press Ctrl-u. More commands can be found from the nano help text, that is opened by pressing Ctrl-g.

![Figure 2.1 Nano editor.](image)

### 2.2.2 Emacs

**Emacs** is a versatile editor that contains large amount of commands, functions and extensions. *Emacs* can be used in two ways: as character based editor like *nano* or through a graphical interface. Using the graphical interface requires that you have FreeNX or X-term connection to CSC. In the character based mode emacs commands are given using Ctrl and Esc keys much like in the case of nano commands. In the graphical interface you can use also mouse based command menus which makes working much more feasible for a new emacs user. To start emacs in CSC environment type:

```
emacs file_name
```

If a graphics enabling connection between CSC and your local computer exists, the command will launch the graphical *emacs* interface. Otherwise the character based mode is used. In the graphical user interface the **File** menu is used to read in and save files. **Edit** menu include commands to copy, cut and paste text. This menu also contains tools to search and replace strings. With the commands in the **Options** menu you can control many properties of *emacs*. You can for example use **Syntax**
highlighting when you are working with a program or a shell script file. You can have several files opened in one emacs session. The **Buffers** menu allows you to choose the buffer to be edited. The **Tools** menu contains miscellaneous commands and tools.

![Graphical interface of emacs editor.](image)

**Figure 2.2:** Graphical interface of emacs editor.

### 2.2.3 Image and PostScript viewers

In some cases application programs, executed at CSC, produce image files instead of text files. If you have a working FreeNX or X-term connection between your local computer and CSC you can use **eog** (Eye of GNOME) and **evince** programs to view image files. **Eog** is a simple image viewing program that can display common image formats like *png*, *bmp*, *jpg* and *tiff*. However **eog** is not able to show *PostScript* (ps) or *pdf* files. For these file types you can use program **evince**.

To view a jpg file, type:

```
eog image_name.jpg
```

and to view a PostScript file, type:

```
evince file_name.ps
```

Post script files can be converted to pdf documents with command **ps2pdf**. This command reads a *PostScript* file and prints out corresponding *pdf* file. For example command:

```
ps2pdf file1.ps
```

Produces a pdf-formatted file *file1.pdf*
2.3 Working with bash shell

bash has several features that make the command line usage easier and more effective. You do not have to always type the whole command line character by character. When you are typing the command, you can go backwards and forward by using the leftwards and rightwards arrow keys. You can delete the character left to the cursor with backspace key. Pressing Ctrl-d or the Delete key deletes the character under the cursor. In addition to these basic command line editing functions you can use the keyboard commands listed in the table below.

Table 2.4: Short-cuts for editing command line

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-f</td>
<td>Move one the cursor character forward in the command line</td>
</tr>
<tr>
<td>Ctrl-b</td>
<td>Move one the cursor character backwards in the command line</td>
</tr>
<tr>
<td>Ctrl-a</td>
<td>Move the cursor to the beginning of the command line</td>
</tr>
<tr>
<td>Ctrl-e</td>
<td>Move the cursor to the end of the command line</td>
</tr>
<tr>
<td>Ctrl-k</td>
<td>Cuts all the characters rightwards from the cursor</td>
</tr>
<tr>
<td>Command</td>
<td>Function</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>Ctrl-y</td>
<td>Paste the characters, cut with Ctrl-k, to the command line</td>
</tr>
<tr>
<td>Ctrl-p</td>
<td>Show the previous command</td>
</tr>
</tbody>
</table>

In bash the executed commands are stored to users home directory into a file called `.bash_history`. To see the full list of recently executed commands give command history. In command line, you can browse the list of previous commands with the **upwards arrow key** and the **downwards arrow key**. Alternatively you can use *Ctrl-p* and *Ctrl-n*. In cases, where you need to give similar commands several times, it is often handy to get one of the previous commands to the command line with the arrow key. Then you can edit just the modifications needed to the old command and execute the modified command by pressing **Return key**.

It's possible to search through the command history. Press *Ctrl-r* then type what you want to search for. You do not need to type a complete search, and the search results will update with each character pressed. To move between the search results use the **upwards arrow key** and the **downwards arrow key**. To run the command press the **Return key**. If you wish to edit the search result before running the command use the **leftwards arrow key**, then edit the command.

### 2.3.2 Automatic Tab completion

If you press *Tab* key the bash shell tries to complete the command or argument you are writing. The completion is done as far as possible using the lists of available commands and files. Using auto-completion is very recommended as it saves the user from typing all command or argument characters and also takes care that command don't get mistyped. For example, let's say we are in a directory where we have two files: *final_research_report_old.pdf* and *final_research_report_new.pdf*.

To open the latter of the files with *evince* program we would need to type command:

```
evince final_research_report_new.pdf
```

This command would require you to type 36 characters. However, by using **Tab completion** you need to type only 6 characters. First type

```
evin
```

and then press the *Tab* key. Normally at CSC, *evince* is the only available program or command that starts with letters *evin* so when the tab completion is executed, it knows to complete the rest of the command:

```
evince
```

Then to define the file name, you can type just the first letter of the file name

```
evicf
```

When you now press *Tab*, the completion process checks what files, starting with *f*, are available. In this case there is two of them and as the beginning of the two file names is the same string, the command can now be completed to:

```
evince final_research_report_
```

Now you just need to type one *n* to the end of file name to distinguish the file from the *old* version,

```
evince final_research_report_n
```
When the Tab key is now pressed again there is only one option that matches the beginning of the argument that has been typed and thus the command is completed and ready to be executed:

```
evince final_research_report_new.pdf
```

2.3.3 **Stopping programs and running programs in background**

In Linux, graphical interfaces and commands that are not interactive once they have started, can be executed as **background processes**. When the command is executed as a background process the command shell does not wait until the command is finished. Instead it remains active and allows user to submit new commands, while the background command gets executed. However, note that in the computing clusters of CSC, the heavy computing tasks should not be executed as background processes, but they should be submitted to the batch queue system.

In normal interactive usage you can launch the command to be executed in the background by adding & character to the end of the command. For example command:

```
eog image1.jpg &
```

would open the **Eye of GNOME** image viewing program (requires X-term or FreeNX connection) to the background so that the command shell could still be used even though the eog program is still running. A background process can be changed to normal, foreground process with command `fg`. When a command and program is running interactively, i.e. the command shell is waiting that the execution finishes, you can terminate the execution by pressing **Ctrl-c**. Another possibility is to halt the program by pressing **Ctrl-z**. When command (or process) is halted, it can be continued with `fg` command or changed to be executed as a background process with command `bg`.

A command that is still running can be terminated with **kill** command. To be able to use `kill`, you need to know the **process identification number** (PID) of the command you want to terminate. You can check your active processes, meaning the commands that you are currently executing, with command `ps`. For example:

```
kayttaj@hippu1:~>ps
PID TTY TIME CMD
385 pts/12 00:00:00 tcsh
2001 pts/12 00:00:00 eog
2003 pts/12 00:00:00 gconfd-2
2203 pts/12 00:00:00 ps
```

By default, the `ps` command shows only those processes that have been launched from the command shell you are currently using. To see all your processes in the server you have logged in, give command:

```
ps -f username
```

Once you have identified the correct process number you can kill the process with command:

```
kill process_ID
```

For example command:

```
kill 2001
```

Would kill the **eog** program, listed in the output of the previous `ps` example. You can kill only processes that are owned by your account. Sometimes, when want to kill a process that is malfunctioning, the normal `kill` command may not able to terminate the process. In those cases you
can try to terminate the process by adding option -9 to the kill command:

```
kill -9 process_ID
```

## 2.4. More commands for managing files

### 2.4.1 Using find to locate files

**Find** command is used to locate a files in the linux file system. The command requires two arguments:

1. the name of the directory where the file is looked for
2. the search condition.

The basic syntax of the command is:

```
find directory search_condition
```

The search condition is normally based to the name of the file(-name value), but you can also use options that refer to dates or access settings. The find command can also have a third argument that defines, what operation is performed to the found files. The default action, that is used if no command argument is given, is -print that prints the path and name of the matching files. Following sample command would look for file called *dataset27.dat* from the current directory. In this case, the file is found from a sub directory *dataset3*.

```
kkayttaj@hippu1:~> find ./ -name dastaset27.txt
./dataset3/dastaset27.txt
```

You can also use *wild cards* in the name search conditions. Note however that in such case you must quote the search condition. Following command locates from your work directory ($WRKDIR) all files that have extension .tmp.

```
find $WRKDIR/ -name "*.tmp"
```

In the last find command examples we use -mtime search condition, that picks files based on their modification date. In $WRKDIR of Hippu, the files that have not been used for 30 days will automatically be removed. With a following command you can check, what files have not been accessed during the last 28 days i.e. the files that are going to be removed in two days:

```
find $WRKDIR -mtime +28
```

Here the +28 means "more than 28 days". In the same ways minus character (-) means less than. So to see what files have been modifies in your current directory less than 24 hours ago, you could use command:

```
find ./ -mtime -1
```

### 2.4.2 File command tells the file type

**File** command evaluates the type of the given file. The syntax of the command is:

```
file file_name
```

The command prints the name of the file and a one line description of the file type. The file
command recognizes most common text file formats, compressed files and Linux executables. It also studies the content of the file and tries to estimate e.g. if a normal text file contains program code or some commonly used data formatting types like XML. Note however, that file often fails to classify correctly application specific files. If the file is a binary file, that is not recognized by the file command, it is reported to be a data file.

In the example below, file types of all the files in the current working directory are listed.

```
kkayttaj@hippu1:~> file ./*
/a.out:         ELF 64-bit MSB MIPS-IV executable, MIPS, version 1
./common.py:   a python script text executable .
/data_old.gz:  gzip compressed data, from Unix
/data.txt:     ASCII text
/intruction.html: HTML document text
/molecule.msv: data
/output4.jpg:  JPEG image data, JFIF standard 1.01
/output4.png:  PNG image data, 640 x 480, 4-bit colormap
/output4.xml:  XML document text
/poster1.pdf:  PDF document, version 1.4
/report.doc:   Microsoft Office Document
```

### 2.4.3 Count rows and characters with wc

Command `wc` (Word Count) is a tool that can be used to count characters (`-m`), words (`-w`) or rows (`-l`) that a Linux text file contains. The most common use of `wc` command is to quickly check the row count of your file:

```
w -l file_name
```

Another common use is checking how many rows output of a command contains. For example following command would give the number of files with extension `.dat` in the current directory.

```
ls *.dat | wc -l
```

### 2.4.4 Comparing two files with diff

`Diff` command can be used to compare two files. `Diff` goes through the files row by row and prints out lines that are not identical. `Diff` is most useful, when you need to compare two nearly identical files like two versions of the same program file. The basic syntax of the command is:

```
diff file1 file2
```

### 2.4.5 Using checksums to verify successful data storage or transfer

Checksums provide a tool to make sure that a data file is fully conserved during storage or copying. The idea behind checksums is an algorithm that calculates a number or a string, based on the content of the file. A checksum string is calculated and stored before the file is moved to a storage media or copied to a new location. Later on, when the data is retrieved from the storage or the
copying process is finished, a new checksum is computed based on the retrieved or copied files. If the new checksum equals to the previously computed one, we can be pretty sure that the data is fully conserved.

One of the most common checksum algorithms is md5 that is often used to verify the correctness of data files. For example many scientific data sets, available for download in the internet, are accompanied by a list of md5 sums. The md5 sum is always a text string 32 characters long. This string has the characteristics of a good checksum: it does not tell anything about the actual content of the source file and any modification to the original file produces a completely different checksum. Other frequently used checksum algorithms include SHA (Secure Hash Algorithm) that is often used in cryptography and CRC (Cyclic redundancy check) that is common in data transport.

In the CSC environment you can generate and check md5 checksums with command \texttt{md5sum}. An md5 checksum for a file is calculated with command:

\texttt{md5sum \textit{file\_name}}

For example:

\texttt{kkayttaj@hippu1:~> md5sum poster1.pdf}
\texttt{cc494699398122a6b6d93a5a69bd2667 poster1.pdf}

You can easily store the checksum to a file by redirecting the output of the command to a new file with > character.

\texttt{md5sum poster1.pdf > poster1.pdf.md5}

The command above stores the checksum and file name to a new file called \texttt{poster1.pdf.md5}.

Checking a set of files against an md5 sum list is done by using option \texttt{-c}.

\texttt{md5sum -c \textit{checksum\_list}}

For example to check validity of file \texttt{poster1.pdf} with the previously created checksum file \texttt{poster1.pdf.md5} could be done with command:

\texttt{kkayttaj@hippu1:~> md5sum -c poster1.pdf.md5}
\texttt{poster1.pdf: OK}

\subsection*{2.4.6 Encrypting files with gpg}

If you need to work with confidential data at CSC, you can use file encryption to increase the security of your data. In normal conditions encrypting files that locate at CSC is not needed. The files can by default be accessed only by the user him/her self. In principle, the the system administrators of CSC are able to read all data at the servers of CSC. In some occasions the administrators may need to check file names and sizes, but the administrator policy of CSC strictly prohibits accessing the contents of customers data files. However, encryption may be reasonable if you for example need to copy the data outside CSC or if encryption is required by the owner of the data.

At CSC, you can use program \texttt{gpg} to encrypt your files. \texttt{Gpg} is frequently used for creating encryption key pairs to protect emails and other data transport. However, in this chapter we demonstrate only how \texttt{pgp} can use used to encrypt individual files.

The basic syntax for encrypting a file with \texttt{gpg} is

\texttt{gpg -c \textit{file\_name}}
The command asks the user to define a password for the file. This password is not, and should not be, in any sense related to your CSC password. After confirming the password the command makes an encrypted copy of the given file. By default the encryption is done with CAST5 algorithm, but several other algorithms can be used too.

To open a gpg encrypted file, give command.

```
gpg gpg_cypterd_file
```

**Gpg example**

Say we have file a *patients.txt* that we want to encrypt. This can be done with command:

```
gpg -c patients.txt
```

When the command is launched, following prompt appears:

```
Enter passphrase:
```

Now you can type in any password for the file. In this case we use following password: y8kleg%a. Once the password is typed, the programs asks you to confirm the password:

```
Repeat passphrase:
```

When the encryption is finished, the we now have two files: the original file and and its' encrypted version that has an extension *.gpg*.

```
kayttaj@hippu1:~> ls -l
-rw-------+ 1 kayttaj csc 1291176 Feb 11 15:57 patients.txt
-rw-------+ 1 kayttaj csc 313848 Feb 11 16:05 patients.txt.gpg
```

Note that in this case the encrypted file is smaller than the original one. Now we can remove the original file.

```
rm patients.txt
```

Later on, for example after copying the file to some other location, you can extract the data with command:

```
gpg patients.txt.gpg
```

The program now asks for the password you used in encryption (in this case: y8kIeg%a). After this you again have two files the encrypted file *patients.txt.gpg* and the original, readable, file *patients.txt*. Note that if you forget the password of your encrypted file, there is no one who can open the file!

### 2.4.7 Managing access permissions of files and directories

Hundreds of users use the computing and storage environments of CSC. To keep the files private and in order, each file and folder in the linux environment of CSC is owned by a certain user account. In linux systems each file has three user categories: *owner*, *group* and *others*. For each of these user categories there is three accession settings: *reading*, *writing* and *execution* permissions.

By default only the owner of the file can read and modify (i.e. write) the files and directories he/she has created. Other users do not have any access permissions to the files. Normally this setting is
good as it keeps your data private. However, if you wish to share some data or execute self written programs the access permissions need to be modified.

You can check the access permissions with command `ls -l`. Let's take a look to the sample file listing that was previously used in the `ls -la example`. In this file listing the characters from second to the tenth character include the information of the access permissions. The same information is shown in the Owner filed in the file manager tool of the Scientist's User interface.

The first three of these accession characters display the permissions of the owner, next three ones display the access permissions for the linux group members and the last three characters for all the other users. Below is a sample output for `ls -la` command:

```bash
total 26914
drwx------+ 3 kkayttaj csc 10 Dec 22 09:12 .
drwxr-xr-x  20 root  root 0 Dec 22 09:12 ..
drwx------+ 42 kkayttaj csc 472 Dec 22 09:07 ..
rwxr-x---+  1 kkayttaj csc 1648 Dec 22 09:01 .cshrc
-rw--------+  1 kkayttaj csc  93 Dec 22 09:01 .my.cnf
-rw--------+  1 kkayttaj csc  48 Dec 22 09:05 Test.txt
-rw--------+  1 kkayttaj csc 878849 Jan 19 2009 input.table
drwxr-xr-x+  2 kkayttaj csc  2 Dec 22 09:11 project1
-rw--------+  1 kkayttaj csc 26432051 Dec 22 09:08 results.out
-rw--------+  1 kkayttaj csc  25 Mar 27 2009 sample.data
-rw--------+  1 kkayttaj csc  49 Mar 27 2009 test.txt
```

In the case of file `Test.txt` the setting is: `rw------`. This means that the owner of the file (kkayttaj) has permission to read (`r`) and write (`w`) to the file. Other users have no permissions for this file. In the case of file `.cshrc` the definition is: `rwxr-x---`. In this case the owner has also execution permissions(`x`) to the file and also the other users that belong to group `csc` have permission to read (`r`) and execute (`x`) the file.

### 2.4.8 Managing access permissions with Scientist's User Interface

In Scientist's User Interface, you can check and modify the access permissions of your files with the file manager tool (`My Files`). First select a file or directory from a file list. Then click the right mouse button, and select **Properties** from the pop up command menu. This opens a new Window that has two tabs: General and Permissions. Selecting the Permissions tab, shows you a list of tick boxes that you can use switch on or off access permissions. You can apply the modification to a whole directory by selecting also the **Apply changes recursively** tick box. Once you have selected right settings, press the **Save** button, and close the Properties window. To check the new permissions of the file or folder you should refresh the file listing. To do this right click the file list, while no files are selected, and select **Refresh** from the pop up command menu.
2.4.9 Managing access permissions in command line usage

In command line usage, access permissions can be modified with command `chmod`. This command needs two arguments: 1. a string that defines what changes are to be done and 2. an argument that defines the target file or directory. In the first argument, you first define the user category: `u` (user i.e. owner), `g` (group) or, `o` (others). Then you define with plus or minus character if you are going to add (+) or remove (-) permissions. Finally you define, what permissions are added or removed. For example, to allow all the group members to read file `Test.txt` you should give command:

```bash
chmod g+r Test.txt
```

You can check the effect with `ls -l` command:

```
kKayttaj@hippu1:~>ls -l Test.txt
-rw-r-----+ 1 kKayttaj csc 48 Dec 22 09:05 Test.txt
```

You can define several user categories and permissions in the same time. Command:

```bash
chmod go+rwx  test.txt
```

would add all access permissions to all users to file test.txt. To remove the permissions you should change the `+` character to `-`

```bash
chmod go-rwx  test.txt
```

Note that by default, changing the permissions of a directory does not change the permissions of the files and subdirectories in the target directory. Thus command:

```bash
chmod g+w project1
```
would not allow other group members to modify the files in directory project1. You can use option
option `-R` to do the same permissions modification `recursively` i.e. to all files and subdirectories in the
target directory
`chmod -R g+w project1`

You can use command `groups` to check which groups you belong to. To see the members of a
specific group, give command
`grep group_name /etc/group`

You can find more information about using unix user groups from chapter 1.4.2

2.5 Commands for data processing

Linux provides plenty of tools to study, filter and modify data files. These tools are frequently
useful when data files need to be modified from one format to another or when specific data needs
to be extracted from large data sets. However plain linux tools have their limitations. If you are
going to do more complex data management, scripting languages like Python and Perl may be more
efficient tools to use. In the case of numerical data many application programs, for example `MatLab`
and `R` provide tools for both complex analysis tasks and automating the analysis.

2.5.1 Grep selects rows that match given string

`Grep` command is used to pick rows from a file. This command is very useful for picking data from
large files, however using grep requires that you are well aware of the contents of the file you are
working with. The basic syntax of grep is:

`grep pattern target_file`

This command goes through the target file and prints out rows that contain the given search pattern.

For example, say we have a file called `authors.txt` that contain following rows:

```
Eeva Pekkanen. Oulu 50
Markus Aho. Turku 50
Pekka Niemi. Tampere 26
Leena Virtanen. Kuopio 32
```

In that case, command:
```
grep Pekka authors.txt
```

Would print two rows:
```
Eeva Pekkanen. Oulu 50
Pekka Niemi. Tampere 26
```

as also `Pekkanen` matches the search string `Pekka`. If we want to use full names as search strings,
including the space character between the names, we would need to embrace the search string
within quotation marks.

grep "Pekka Niemi" authors.txt

Would print out just:

Pekka Niemi. Tampere 26

An inverse selection could be done using option -v. For example command

grep -v Pekka authors.txt

would print out:

Markus Aho. Turku 50
Leena Virtanen. Kuopio 32

By default grep is case sensitive and thus command:

grep pekka authors.txt

would not result any hits. With option -i, grep ignores the difference between upper and lower case letters and thus command

grep -i pekka authors.txt

Would again print rows

Eeva Pekkanen. Oulu 50
Pekka Niemi. Tampere 2

Grep interprets the given search string as a regular expression. This means that certain characters have special meaning and they are interpreted as part of the regular expression string. The same regular expression syntax is used with awk and sed commands too. For example:

- dot (.) is used to define any single character.
- ^ means the beginning of a line
- $ means the end of a line
- \[ \] means any of the characters inside the brackets. For example [abc] would match a,b or c.
- \[^ ] means match any character, except the characters inside the brackets. For example \[^abc]\ would select all rows that contain also other characters than just a,b and c.
- * match zero or more of the preceding character or expression
- \{n,m\} match n to m occurrences of the preceding character or expression

For example command

grep "i." authors.txt

Would print following two rows as the search criteria is in this case: i and any other character.

Pekka Niemi. Tampere 26
In these kind of situations you can use backslash character (\) to define that the following character should not be interpreted as part of a regular expression. Thus command:
    grep "i\." authors.txt

Would now return just one row.

Grep is often used to filter the output of other commands. For example, to so see what files in the current directory are from year 2010 we could use command `ls -l` and pipe it to `grep`. With `grep` we could select from the output only those rows that include string 2010. However, some file names may contain string 2010 ever though they are not dating from 2010. To make the filtering more effective we could use quotation marks to include the space before and after the year number to the search string.
    ls -l | grep " 2010 "

In addition to normal `grep` command, there exists several other grep like tools. For example `zgrep` or `bzgrep` can be used to search rows from files compressed with `gzip` or `bzip2`. In Hippu, there is also a tool called `agrep` that can be used to select rows that can have certain amount of mismatches to the search string.

### 2.5.2 Using sed to select rows by row number

The stream editor `sed` can be used to select and modify rows from a text file. Many of the `grep` operations, described above could be done with `sed` too. The `sed` command is discussed a bit more in detail in the chapter 2.5.8 Replacing characters and strings. Here we show only how `sed` can be used to select a certain row from a file. For example the third row of the file `authors.txt` could be selected with command:
    sed -n "3"p authors.txt

In this case the result would be:

Pekka Niemi. Tampere 26

This row picking method is handy in loops. For example the following loop would print out three rows from the example file:

```bash
for i in 4 3 2
do
    set row=(`sed -n "$i"p authors.txt `)
    echo "Row $i looks like: $row"
done
```
2.5.3 Simple column selection with cut

The *cut* command provides a simple tool to select character *regions* (-c ) or *columns* (-f) from a linux text file. When selecting by character numbers the syntax of the command is:

```
cut -c start-end
```

For example command:

```
cut -c2-5 authors.txt
```

Would print text from second to the fifth character from each row of the *authors.txt* file. Assuming this is the same file that was used in the grep examples above, the output would look like:

```
eva
arku
ekka
eena
```

With option -f columns can be printed. By default *tabulator* is used and the column delimiter, but other delimiters can be defined too with option -d . For example, if we would like to select first and third columns (-f1,3) from the file authors.txt we should define that space is used as the delimiter (-d " ")

```
kayttaj@hippu2:~> cut -d " " -f1,3 authors.txt
```

```
Eeva Oulu
Markus Turku
Pekka Tampere
Leena Kuopio
```

2.5.4 Using awk to work with columns

*Awk* is a scripting language that can be used to process text files in linux environments. *Awk* is a rather old programming language, and nowadays more modern scripting tools are normally used for creating data management scripts. However, due to very simple syntax *awk* is still frequently used to perform simple filtering and editing tasks. In this guide we don't provide general overview to *awk* programming. Instead we show some examples how *awk* can be used to work with files or as a part of linux command pipeline. In the simple command line usage, the basic syntax of *awk* is:

```
awk -F "field delimiter" 'statement {command1|command2}...' infile.dat.
```

*Awk* can be used to process the input data as columns. By default *awk* uses both space and tab characters as column delimiters ( note that unlike *cut* command, *awk* interprets all successive spaces as one single column separator). The columns can be accessed in awk command with variable names $1 (the first column), $2 (second column), $3 (third column) etc. Expression $0 can be used to refer "all the columns". For example following command prints the first and third columns of file authors.txt.

```
kayttaj@hippu2:~> awk '{print $1 $3}' authors.txt
```

```
EevaOulu
MarkusTurku
PekkaTampere
```
Leena Kuopio

In the example above the fields are not separated in the output. To do that you should add space or tabulator character (\t) to the print statement. You can also add other text to the output. Then next example adds text and changes the order of columns.

```
awk '{print "Location: "$3"\t Name: "$1" "$2"}' authors.txt
```

```
Location: Oulu   Name: Eeva Pekkanen.
Location: Turku Name: Markus Aho.
Location: Tampere Name: Pekka Niemi.
Location: Kuopio Name: Leena Virtanen.
```

A more sophisticated way to do the modification above would be to use `printf` command instead of `print`. The syntax of the `awk printf` command is similar to that of the Linux command `printf` discussed in chapter 2.7.5.

```
awk '{printf "Location:%s\tName: %s %s \n", $3, $1, $2}' authors.txt
```

One of the nice features of `awk` is that you can use conditional statements in the command. You can compare both strings and numbers. For example following command would print data only when the third column is "Tampere"

```
awk '{if ( $3 == "Tampere") print "Location:"$3"\t Name: "$1" "$2"}' authors.txt
```

```
Location: Tampere Name: Pekka Niemi.
```

Alternatively, the condition could be defined in the statement part before the `awk` commands,

```
awk ' $3 == "Tampere" {print "Location:"$3"\t Name: "$1" "$2"}' authors.txt
```

If columns contain numerical values `awk` can also do numerical operations. For example, let's use `awk` to calculate how much disk space jpg image files take in the current folder. We can first list the jpg files with command:

```
ls -l *.jpg
```

```
-kayttaj@hippu2:~> ls -l *.jpg
-rw-------+ 1 kkayttaj csc  85112 Dec 19 2002 Image_2.jpg
-rw-r--r--+ 1 kkayttaj csc 105542 Nov  6 2006 Screen.jpg
-rw-r--r--+ 1 kkayttaj csc 167598 May 15 2008 casa1.jpg
-rw-------+ 1 kkayttaj csc  550031 Mar 25 2008 hexa.jpg
-rw-------+ 1 kkayttaj csc  869534 Dec 23 2005 img.jpg
-rw-------+ 1 kkayttaj csc  19159 Jul 23 2003 kuva.jpg
-rw-r-xr-x+ 1 kkayttaj csc  433268 Dec 23 2005 plot3.jpg
-rw-------+ 1 kkayttaj csc 1591444 Jan 26 12:27 copy.jpg
```
To sum up the values of the file sizes in the fifth column, we pipe the output to awk.

```
kkayttaj@hippu2:~> ls -l *.jpg | awk '{total+=$5} END {print "JPG files in total: "total}'
JPG files in total: 3821688
```

Note that in the `awk` command above the `total` is a variable name used by `awk`. The `END` defines that the following command is executed only when all input lines are processed. In the second `awk` computing example, we calculate and display a cumulative sum and average of ages from file `authors.txt`.

```
awk '{ tot_age = tot_age + $4 }{ aver=tot_age/NR}{ print $0 " cumulative age sum:" tot_age " average:" aver}' authors.txt
```

Here we use the automatic `NR` row count variable to calculate average of the age. This command prints out:

```
Eeva Pekkanen. Oulu 50 cumulative age sum:50 average:50
Markus Aho. Turku 50 cumulative age sum:100 average:50
Pekka Niemi. Tampere 26 cumulative age sum:126 average:42
```

### 2.5.5 Using sort to order rows

The `sort` command can be used to order rows of text file on input data into alphabetical or numerical order. The syntax of the `sort` command is simple:

```
sort -options files_to_sort
```

By default the `sort` command orders the rows of the file alphabetically using case sensitive sorting. With option `-i`, case insensitive sorting is performed. If the data contains columns, separated by space or tabulator character you can ask the sort command to use certain column (or columns) for ordering the data. The columns to be used can be selected with option `-k column number`. For example to sort the data in the file `authors.txt` (the example file used previously in this chapter), according to the family names that locate in the second column, you could use command:

```
sort -k 2 authors.txt
```

That would print out:

```
Markus Aho. Turku 50
Pekka Niemi. Tampere 26
Eeva Pekkanen. Oulu 50
Leena Virtanen. Kuopio 32
```

You can also define other columns separators with option `-t separator`. Note that by default `sort` orders also numbers alphabetically. Option `-n` makes `sort` to use numerical order instead. Option `-r` can be used to reverse the order. Numerical sorting according to the fourth column can now be done with command

```
sort -k 4 -n -r authors.txt
```
sort -k4 -n authors.txt

And it would result following result:
Pekka Niemi. Tampere 26
Leena Virtanen. Kuopio 32
Eeva Pekkanen. Oulu 50
Markus Aho. Turku 50

You can also define that several columns will be used for sorting. For example the following command would use the numerical order of the fourth column as the primary sorting criteria (-k4n). In the cases where several rows have the same value in the fourth column, the alphabetic order is used as the secondary sorting criteria (-k2)
sort -k4n -k2 authors.txt

In this case, the command would result:
Pekka Niemi. Tampere 26
Leena Virtanen. Kuopio 32
Markus Aho. Turku 50
Eeva Pekkanen. Oulu 50

The sorted data can be saved to a new file either by using option -o or redirecting the standard output to a file with > character. For example the two commands below could both be used to create the sorted version of authors.txt file.
sort -k4n -k2 -o authors_sorted.txt authors.txt
sort -k4n -k2 authors.txt > authors_sorted.txt

2.5.6 Removing duplicate rows with uniq

Command uniq is frequently used in combination with sort command to remove redundancy from a data set. The uniq command checks if two successive rows in the input file or standard input are identical. By default uniq removes the successive duplicate rows. Note that uniq does not check if an identical row exists elsewhere in the input. Because of that, the data is normally sorted before applying the uniq command. As an example, say we have a file called names that contains following rows:
Pekka
Pekka
Leena
Markus
Eeva
Leena

Running command:
uniq names

will give following output:
In the output, row "Leena" still occurs twice as the two identical names were not in the successive rows. The situation can be fixed by sorting the rows before uniq is applied. For example, command:

```
sort names | uniq
```

would print following output:

```
Eeva
Leena
Markus
Pekka
```

It is often useful to know, how many identical rows were found. This information can be added to the output of `uniq` with option `-c`. For example, command:

```
sort names | uniq -c
```

Gives following output:

```
 1 Eeva
 2 Leena
 1 Markus
 2 Pekka
```

Note that also space and tabulator characters are used when `uniq` compares the rows. Thus a row containing "Leena" is not identical to a row containing "Leena ". By default `uniq` is case sensitive, but using option `-i` makes uniq to ignore the cases and consider "leena" and "Leena" to be identical. However, note that if you use case insensitive `uniq`, you may need to do also the sorting in case insensitive mode using command `sort -f`.

### 2.5.7 Replacing characters with tr

The `tr` command can be used to replace individual characters. `Tr` reads the standard input and writes the translated data to the standard output. The syntax of `tr` command is

```
tr search_character replacement_character
```

For example, dots (.) in the file `authors.txt` could be replaced with commas (,) with command:

```
tr "." "," < authors.txt
```

would print out:

```
Eeva Pekkanen, Oulu 50
Markus Aho, Turku 50
```
Pekka Niemi, Tampere 26
Leena Virtanen, Kuopio 32

Note that the command above does not modify the original input file in any way. In the examples here, the results of \texttt{tr} command are printed to the screen. When \texttt{tr} is used to modify large files, the output should of cause be guided to a new file instead of screen. For example:

\texttt{tr "." "," < authors.txt > authors_mod.txt}

If you would like to remove the dots, you could use the option \texttt{-d} with just one character set, telling what characters are to be removed:

\texttt{tr -d "." < authors.txt}

Note that the \texttt{tr} command translates individual characters, not words. Thus command \texttt{tr "Oulu" "Turku"} would not translate word "Oulu" to "Turku". Instead it would do following character conversions to all the text O->T, l->r, u->k. The \texttt{tr} command can do translations with special characters like tabulator (\texttt{\textbackslash t}) and newline (\texttt{\textbackslash n}) and with predefined character sets like all lower case letters ([:lower:]) and all upper case letters ([:upper:]). For example command:

\texttt{tr "." \"\n" < authors.txt}

would replace the dots with line breaks in the file \texttt{authors.txt}:

\begin{verbatim}
Eeva Pekkanen
   Oulu 50
Markus Aho
   Turku 50
Pekka Niemi
   Tampere 26
Leena Virtanen
   Kuopio 32
\end{verbatim}

Modifying the file so that all the text is written with upper case letters could be done with:

\texttt{tr "[:lower:]" "[:upper:]" < authors.txt}

Resulting:

\begin{verbatim}
EEVA PEKKANEN, OULU 50
MARKUS AHO, TURKU 50
PEKKA NIEMI, TAMPERE 26
LEENA VIRTANEN, KUOPIO 32
\end{verbatim}

Check the manual page of \texttt{tr} (\textit{man tr}) to see the list of all special characters and character sets that \texttt{tr} can use.
2.5.8 Replacing words and strings with sed

Replacing text strings can be done with `sed`. `Sed` is a stream editor that can be used for many different text processing operations. `Sed` reads string data, either from a file or piped from another command, and does the edition operations defined by the user and then prints the edited string to the standard output. `Sed` is a very powerful, though a bit hard to learn, tool for automatic text editing. In this guide we do not provide a general introduction sed as even a modest overview would take several pages. Instead we show few practical examples how sed can be used. Sed is commonly used with syntax:

```
sed -e sed_script input-file
```

The `sed script` is typically short formulation that defines what kind of editing should be done. For example to do a replacement operation we could use the `sed` operation `s/"search string"/"replacement string"/g`. In this formulation `s` means search and replace operation. The `g` in the end of the formulation means that the replacement operation is global i.e. all matching strings will be replaced. You could replace, for example, just the first or second occurrence of the search string on a row by using number 1 or 2 instead of `g`. For example the operation where Oulu is replaced by Turku in the file `authors.txt` can be done with command:

```
sed -e s/"Oulu"/"Turku"/g authors.txt
```

And the result would now look like:

```
Eeva Pekkanen. Turku 50
Markus Aho. Turku 50
Pekka Niemi. Tampere 26
Leena Virtanen. Kuopio 32
```

Just like `grep` command, `sed` interprets the given search string as a regular expression. This means that some characters, are interpreted as special regular expression defining characters. For example dot (.) is used to define any single character. Thus `sed` command `s/"."/"/g` would change not just dots but all characters to commas (,). In these kind of situations you can use backslash character (\) to tell to the `sed` command that the following character should not be interpreted as part of a regular expression. On the other hand using the regular expressions can make `sed` very effective. Below is some regular expression examples applied the `authors.txt` file with `sed`.

**Example 1.** Using $ character to define the end of a line (note the single quotation marks (’ ) that prevent the $ character to be interpreted as bash variable indicator):

```
sed -e s/’0$’/”1 changed”/g authors.txt
```

Prints out:

```
Eeva Pekkanen. Oulu 51 changed
Markus Aho. Turku 51 changed
Pekka Niemi. Tampere 26
Leena Virtanen. Kuopio 32
```
**Example 2.** Using dot (.) to define any single character:

```
sed -e s/"e.a"/"EXA"/g authors.txt
```

Prints: out:
EEXA Pekkanen. Oulu 50
Markus Aho. Turku 50
Pekka Niemi. Tampere 26
LeEXA Virtanen. Kuopio 32

**Example 3.** Using "^" to define beginning of a line.

```
sed -e s/"^P"/"START:P"/g authors.txt
```

Prints out:
Eeva Pekkanen. Oulu 50
Markus Aho. Turku 50
START:Pekka Niemi. Tampere 26
Leena Virtanen. Kuopio 32

### 2.6 Packing and compression tools

When large data sets are stored at CSC or transported over then net it is usually reasonable to pack and compress the data. Compressed files require less storage space and are thus also faster and easier to move from one server to another. In this chapter we provide introduction to **tar**, **gzip**, **zip**, and **bzip2** tools that are frequently used for packing and compression. Data packing and compression makes file management easier, but it also requires computing time: compressing a gigabyte size file can already take several minutes and when several terabytes needs to be compressed, the task can easily require overnight computing.

#### Table 2.5: Frequently used packing and compression tools and corresponding file name extensions.

<table>
<thead>
<tr>
<th>Type</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>zip archive file</td>
<td>zip, .ZIP or .Z</td>
</tr>
<tr>
<td>file compressed with gzip</td>
<td>.gz</td>
</tr>
<tr>
<td>file compressed with bzip2</td>
<td>.bz2 or .bz</td>
</tr>
<tr>
<td>tar archive file</td>
<td>.tar</td>
</tr>
<tr>
<td>tar archive file that is compressed with gzip</td>
<td>.tar.gz or .tgz</td>
</tr>
<tr>
<td>tar archive file that is compressed with bzip2</td>
<td>.tar.bz2 or .tbz2</td>
</tr>
</tbody>
</table>

#### 2.6.1 Tar and gtar: packing several files into one file

The **tar** command was originally developed to make back up copies of linux directory structures with tape drives. However nowadays **tar**, and its GNU version **gtar**, are mostly used for data
packing within the normal disk environment. Tar commands pack all the files in a given directory or file list to a single file, often called as a *tar archive file* or a *tarball*. The main reason for creating tar files is that moving and storing of a one large file is easier that trying to manage large set of separate files.

By default, tar commands do not compress the data. This means that the size of the tar archive file is the same as the sum of the sizes of packed files. If data compression is needed, you can use compression tools *gzip* or *bzip2* with *tar*.

*Tar* and *gtar* are compatible with each other; if you pack your data with *tar* you can unpack it with *gtar* and vice versa. Normally *tar* or *gtar* is found from any unix, linux or MacOSX machine. In Windows machines you can use for example *7zip* program to manage tar archive files.

The syntax of *tar* command is:

```
tar options tar-file target
```

For example, say we have a directory called *project_3* that contains nine files called: *sample1.txt*, *sample2.txt* ... *sample9.txt*. To see the content of the directory we in this case use command *ls -lh*.

```
testuser@hippu1:/wrk/testuser> ls -lh project_3/
total 53M
-rw-r--r-- 1 testuser csc 16M Nov  9 10:41 sample1.txt
-rw-r--r-- 1 testuser csc 16M Nov  9 10:41 sample2.txt
-rw-r--r-- 1 testuser csc 1.3M Nov  9 10:41 sample3.txt
-rw-r--r-- 1 testuser csc 1.9M Nov  9 10:41 sample4.txt
-rw-r--r-- 1 testuser csc 1.9M Nov  9 10:41 sample5.txt
-rw-r--r-- 1 testuser csc 3.7M Nov  9 10:41 sample6.txt
-rw-r--r-- 1 testuser csc 4.0M Nov  9 10:42 sample7.txt
-rw-r--r-- 1 testuser csc 3.9M Nov  9 10:42 sample8.txt
-rw-r--r-- 1 testuser csc 3.9M Nov  9 10:42 sample9.txt
```

You can pack all the files in the *project_3* directory to a tar file called *project_3.tar* with command

```
testuser@hippu1:/wrk/testuser> tar cvf project_3.tar project_3
```

The command above creates a new tar archive file, *project_3.tar*, that contains all the files from directory *project_3*. Note that the command do not modify or remove the original files in the source directory.

```
testuser@hippu1:/wrk/testuser> ls -lh
-rw-r-xr-x 2 testuser csc 52M Nov  9 10:46 project_3
```

The *tar* command does not require that you assign certain extension, like *.tar*, for your archive files. However, applying the commonly used file name extensions will help you and other users to select right commands for unpacking the data later on. The tar archive file can now be easily moved to another directory or machine and then unpacked with command:

```
tar xvf project_3.tar
```

This command creates directory *project_3* that contains the same files as the original directory. Note that if *tar xvf* command extracts a file that already exist in the file system, the existing file will be overwritten by the extracted file. This causes a potential danger: a newer version of a file will be lost if an older version of the same file exists in the tar archive that is unpacked. The first tar option letter defines the operation to be performed. In the tar examples above options *c*(create) and
x(extract) were used. Below is listed the available tar operations:

<table>
<thead>
<tr>
<th>Tar option</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>append tar files to an archive</td>
</tr>
<tr>
<td>c</td>
<td>create a new tar archive</td>
</tr>
<tr>
<td>d</td>
<td>find differences between archive and file system</td>
</tr>
<tr>
<td>r</td>
<td>append files to the end of an archive</td>
</tr>
<tr>
<td>t</td>
<td>list the contents of an archive</td>
</tr>
<tr>
<td>u</td>
<td>only append files that are newer than the existing in archive</td>
</tr>
<tr>
<td>x</td>
<td>extract files from a tar file</td>
</tr>
<tr>
<td>--delete</td>
<td>delete from the archive</td>
</tr>
</tbody>
</table>

In addition to the operation defining options above, option f is nearly always used as it defines that the name of the tar file to read from or write to is given as the first argument. Below are some frequently used tar options:

<table>
<thead>
<tr>
<th>Tar option</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>use given file name as the source or target tar file</td>
</tr>
<tr>
<td>j</td>
<td>use bzip2 compression/decompression while packing or unpacking the tar file</td>
</tr>
<tr>
<td>v</td>
<td>list processed files to the screen while processing</td>
</tr>
<tr>
<td>z</td>
<td>use gzip compression/decompression while packing or unpacking the tar file</td>
</tr>
</tbody>
</table>

By adding option z to the packing command you could compress the files to be archived:

```
tar cvzf project_3.tar.gz project_3
```

In our sample case the size of the uncompressed archive file is 52 MB but the compressed file is only 15 MB.

```
testuser@hippu1:/wrk/testuser> ls -lh
drwxr-xr-x 2 testuser csc 11 Nov 9 10:44 project_3
-rw-r--r-- 1 testuser csc 52M Nov 9 10:46 project_3.tar
-rw-r--r-- 1 testuser csc 15M Nov 9 10:56 project_3.tar.gz
```

Listing the content of a tar archive file can be done with options tf.

```
testuser@hippu1:/wrk/testuser> tar tf project_3.tar
project_3/
project_3/sample3.txt
project_3/sample4.txt
project_3/sample5.txt
project_3/sample2.txt
project_3/sample1.txt
project_3/sample6.txt
project_3/sample8.txt
```
project_3/sample9.txt
project_3/sample7.txt

You can also retrieve just one file from the tar archive by adding the file name to the data extraction command. For example to extract just file sample2.txt from project_3.tar.gz we could use command

tar xzvf project_3.tar.gz project_3/sample2.txt

### 2.6.2 Compressing files

Compressing saves storage space, but it may take a lot of time. Therefore, it should be used mainly for large files that will need to be stored for a long time or moved to another location. There are numerous algorithms and software tools available for data compression. Here we focus to the three tools that are most frequently used in linux systems: **gzip**, **bzip2** and **zip**. What is common to all these tools is that they do the compression without data loss i.e. when the files are uncompressed so that the later on uncompressed files will be 100 % identical to the original files. Typically, a text file compressed with one of these tools is 20-60% smaller that the original file. However, the compressibility of a file depends heavily on the content of the file.

#### 2.6.2.1 Gzip and gunzip

**Gzip** is probably the most commonly used packing tool in unix and linux machines. It uses Lempel-Ziv coding (LZ77) for packing the data. Gzip was already briefly mentioned in the tar chapter above, however the command can be used also as a totally separate tool. The normal usage of **gzip** is straight forward. To compress the file, give command:

```
gzip file_name
```

Running this command creates a compressed file and names it using the original file name and extension .gz. When the compression is ready the original file is removed. If you want to preserve the original file, you can use the command with syntax:

```
gzip < file_name > file_name.gz
```

Decompression of a gzipped file is done with command **gunzip**. In addition to **gzip** compressed files, gunzip can decompress also files compressed with **zip** command. The basic syntax of **gunzip** is analogous to the compressing command:

```
gunzip file_name.gz
```

The command above removes the compressed file when decompression is ready. If you wish the keep the compressed file too you can use syntax:

```
gunzip < file_name.gz > file_name
```

**gzip** and **gunzip** have several command line options that are not discussed here. Use the **man** command to see the full list of available options:

```
man gzip
man gunzip
```

**gzip example**

For example lets assume we are in the $WRKDIR directory of Hippu and we have there just one
file called *my_data.dat*. Let's first check the size of the file with command `ls -lh`

```
testuser@hippu1:/fs/metawrk/wrk/testuser/> ls -lh
total 1.5G
-rw-r--r--+ 1 testuser csc 1.5G Nov 4 13:07 my_data.dat
```

The listing tells us that the size of the file is approximately 1.5 GB. Next we compress the data with `gzip` and then check the file size again.

```
testuser@hippu1:/fs/metawrk/wrk/testuser/> gzip my_data.dat
```

```
testuser@hippu1:/fs/metawrk/wrk/testuser/> ls -lh
total 834M
-rw-r--r--+ 1 testuser csc 833M Nov 4 13:07 my_data.dat.gz
```

Data compression and decompression requires a significant amount of computing. Compressing a file of few gigabytes can take several minutes. However, as a result we now have a compressed file that requires only 833MB of disk space (55% of the original size). Next we decompress the file:

```
testuser@hippu1:/fs/metawrk/wrk/testuser/> gunzip my_data.dat.gz
```

```
testuser@hippu1:/fs/metawrk/wrk/testuser/> ls -lh
total 1.5G
-rw-r--r--+ 1 testuser csc 1.5G Nov 4 13:07 my_data.dat
```

The file listing shows that the compressed file has now disappeared and the original file is now available again.

### 2.6.2.2 bzip2 and bunzip2

Bzip2 is a compression program that is used very similarly compared to `gzip`. The main difference between the two programs is that `bzip2` uses Burrows-Wheeler block sorting text compression algorithm combined with Huffman coding instead of the LZ77 algorithm used in `gzip`. The compression algorithm of `bzip2` produces more effective packing than `gzip`. However, computing the `bzip2` compression normally takes longer than `gzip` compression. Note that a file compressed with `bzip2` can not be uncompressed with `gunzip` and vice versa. The command-line usage of `bzip2` is very similar to that of `gzip`, but all the command line options are not identical. The basic compression syntax is:

```
bzip2 file_name
```

or

```
bzip2 < file_name > file_name.bz2
```

Similarly, the decompression can be done with command:

```
bunzip2 file_name.bz2
```

or

```
bunzip2 < file_name.bz2 > file_name
```

In addition to the standard `bzip2` and `bunzip2` programs, you can also use the parallel versions of `bzip2` command: `pbzip2` and `pbunzip`. When these commands are used, the user must use option `-p` to define the number of processor cores to be used. For example, compressing file `my_data.dat` using four cores can be done with command:

```
pbzip2 -p4 my_data.dat
```
Similarly, to decompress the file with two cores you can use command:

```
punbzip2 -p2 my_data.dat.bz2
```

The `pbzip2` and `pbunzip2` commands scale well for small core numbers. Already with two core `pbzip2` is abut as fast as `gzip`. The number of processors used does not affect to the actual result file. Thus a file that has been compressed with parallel `pbzip2` can be uncompressed with normal `bunzip2` command and vice versa.

### 2.6.2.3 Zip and unzip: combined compression and file packing tool

Zip program can be used for both packing and compressing files. Given a list of files or a directory the `zip` command packs and compresses all the files to a single zip archive file. So in principle `zip` is analogous to combination of `tar` and `gzip` commands. Later on the whole archive can be unpacked or just certain files can be retrieved from the archive. The basic syntax of `zip` command is:

```
zip -options archive_file_name source_name
```

The source name can be a file, directory or a list of files that will be packed to the zip archive file. If the given archive file name already exists `zip` will replace the existing old files in the zip archive or add files if corresponding file names do not yet exist in the archive. Note that unlike `tar` command, `zip` does by default pack all the files and sub folders to the zip archive file. Option `-r` is needed to recursively pack all files and subfolders from a given directory to the zip archive. Below is listed some commonly used `zip` command options.

**Table 2.8: Options for `zip` command**

<table>
<thead>
<tr>
<th>zip option</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d</td>
<td>remove file from archive</td>
</tr>
<tr>
<td>-e</td>
<td>encrypt the archive with a password</td>
</tr>
<tr>
<td>-f</td>
<td>Replace (freshen) an existing entry in the zip archive only if it has been modified more recently than the version already in the zip archive</td>
</tr>
<tr>
<td>-l</td>
<td>Translate the Unix end-of-line character LF into the MSDOS convention CR LF.</td>
</tr>
<tr>
<td>-ll</td>
<td>Translate the MSDOS end-of-line CR LF into Unix LF.</td>
</tr>
<tr>
<td>-@</td>
<td>Take the list of input files from standard input. Only one file name per line.</td>
</tr>
<tr>
<td>-r</td>
<td>Travel the directory structure recursively.</td>
</tr>
</tbody>
</table>

Building the zip archive does not affect to the original files. Note that by default `zip` command will add the `.zip` extension to the archive file name, if it is not already included to the archive file name.

Zip archives can be extracted and studied with command `unzip`. To extract files from a zip archive, give command:

```
unzip archive_file_name
```

To just see the files included to the zip archive give command

```
unzip -l archive_file_name
```

You can also extract just one file from an archive with command

```
unzip archive_file_name file_name
```
Table 2.9: Options for unzip command.

<table>
<thead>
<tr>
<th>unzip option</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-f</td>
<td>Freshen existing files, i.e., extract only those files that already exist on disk and that are newer than the disk copies.</td>
</tr>
<tr>
<td>-l</td>
<td>List the content of archive file</td>
</tr>
<tr>
<td>-u</td>
<td>Update existing files and create new ones if needed.</td>
</tr>
<tr>
<td>-o</td>
<td>Overwrite existing files without prompting. This is a dangerous option, so use it with care.</td>
</tr>
<tr>
<td>-P</td>
<td>Password use password to decrypt encrypted zip file.</td>
</tr>
</tbody>
</table>

**Zip example**

For example packing and compressing the sample directory `project_3` that contains files `sample1.txt` - `sample9.txt` (the same example that was used in the `tar` chapter) can be done with command:

```
testuser@hippu2:/wrk/testuser> zip -r project_3.zip project_3
```

```
adding: project_3/ (stored 0%)
adding: project_3/sample5.txt (deflated 71%)
adding: project_3/sample2.txt (deflated 72%)
adding: project_3/sample3.txt (deflated 70%)
adding: project_3/sample4.txt (deflated 71%)
adding: project_3/sample9.txt (deflated 71%)
adding: project_3/sample7.txt (deflated 71%)
adding: project_3/sample1.txt (deflated 73%)
adding: project_3/sample6.txt (deflated 72%)
adding: project_3/sample8.txt (deflated 71%)
```

Note that if you use the same zip command without the `-r` option, the archive file does not include the sample files in the directory but just the directory name. However, in this case the `project_3` directory does not include any subdirectories, so you could do the packing also with command:

```
testuser@hippu2:/wrk/testuser> zip project_3.zip project_3/*
```

```
adding: project_3/sample1.txt (deflated 73%)
adding: project_3/sample2.txt (deflated 72%)
adding: project_3/sample3.txt (deflated 70%)
adding: project_3/sample4.txt (deflated 71%)
adding: project_3/sample5.txt (deflated 71%)
adding: project_3/sample6.txt (deflated 72%)
adding: project_3/sample7.txt (deflated 71%)
adding: project_3/sample8.txt (deflated 71%)
adding: project_3/sample9.txt (deflated 71%)
```

In the same way, you could later on add a new file to an existing zip archive. For example:

```
testuser@hippu2:/wrk/testuser> cp sample10.txt project_3/
testuser@hippu2:/wrk/testuser> zip project_3.zip project_3/sample10.txt
```

```
adding: project_3/sample10.txt (deflated 69%)
```
You can check the content of the zip archive with `unzip` and `-l` option.

```
testuser@hippu2:/wrk/testuser> unzip -l project_3.zip
Archive: project_3.zip
Length  Date   Time    Name
-------- -------------- --------------- ---------------
16662202 11-09-09 10:41 project_3/sample1.txt
16397702 11-09-09 10:41 project_3/sample2.txt
1303352  11-09-09 10:41 project_3/sample3.txt
1925824  11-09-09 10:41 project_3/sample4.txt
1989706  11-09-09 10:41 project_3/sample5.txt
3813333  11-09-09 10:42 project_3/sample6.txt
4176523  11-09-09 10:42 project_3/sample7.txt
4056375  11-09-09 10:42 project_3/sample8.txt
4085713  11-09-09 10:42 project_3/sample9.txt
6541306  11-10-09 13:07 project_3/sample10.txt
-------  -------  -------  -------
60952036 10 files
```

To extract just file `sample3.txt` from the zip archive file, give command:

```
testuser@hippu2:/wrk/testuser> unzip project_3.zip
Archive: project_3.zip
inflating: project_3/sample3.txt
```

To extract all the files from the archive, give command:

```
testuser@hippu2:/wrk/testuser> unzip project_3.zip
Archive: project_3.zip
inflating: project_3/sample1.txt
inflating: project_3/sample2.txt
inflating: project_3/sample3.txt
inflating: project_3/sample4.txt
inflating: project_3/sample5.txt
inflating: project_3/sample6.txt
inflating: project_3/sample7.txt
inflating: project_3/sample8.txt
inflating: project_3/sample9.txt
inflating: project_3/sample10.txt
```

### 2.7 Linux bash scripts

One way of utilizing the flexibility of Linux is using command scripts. A command script is simply a file, which contains a set of normal Linux commands that the command shell will perform automatically in the given order. Compared to real programming languages, like Python, Perl, or C,
programming with Linux (bash, tcsh, csh or sh) is computationally rather ineffective. However, often handy Linux scripts can be constructed in few minutes. You do not have to know too much about command scripting to be able to write simple programs that save a lot of work.

2.7.1 Constructing a script file
A script file is a simple text file that can be constructed with normal text editors like nano, emacs or vi. To create a new script file, type for example:

```
nano my_test.script
```

A script file usually starts with a command line which defines the command shell to be used. In guide we use bash shell, which is the default command shell at CSC. The bash defining row is:

```
#!/bin/bash
```

After that you add the Linux commands, you wish to perform. In practice, just type to the file the commands, that you would normally use to do the task in active command shell. For example, the following script can be used to create a sub directory "mapfiles" and copy all .map files into this new directory:

```
#!/bin/bash
mkdir mapfiles
cp *.map mapfiles/
```

If a line in the script starts with a # mark, it will be skipped, and rest of the line is considered as a comment (with the exception of the first line that start with #!).

```
#!/bin/bash
# This is a comment line that is not executed
mkdir mapfiles
cp *.map mapfiles/
```

After saving the script file and closing the editor, you can perform the commands in the script file by giving a command:

```
source my_test.script
```

Optionally you could give execution permissions for your script file with command:

```
chmod u+x my_test.script
```

And then execute the script with command:

```
./my_test.script
```

2.7.2 Variables and arrays
You can use variables, loops and conditional statements in the scripts. Variables can be set with syntax:

```
variable=value
```

Note that there is no spaces around the equal-to sign.
Variables are recalled with `$` sign:

```
$variable
```

or

```
${variable}
```

For example, command:

```
echo $variable
```

writes the value of variable to the output. Note that in bash scripts the variables are considered to be either strings (i.e. text) or integers. This means that decimal numbers can't be used in bash scripts for mathematical operations.

Example of using string variables

```
[kkmattil@tunturi ~]$ name=Veikko
[kkmattil@tunturi ~]$ familyname=Salo
[kkmattil@tunturi ~]$ address="CSC Espoo"
[kkmattil@tunturi ~]$ echo "Person: ${name} ${familyname} works at ${address}".
Person: Veikko Salo works at CSC Espoo.
```

For integer variables, you can do simple arithmetics with syntax: `((expression))`. Commonly used arithmetic operations are listed in table 2.10

**Table 2.10** Basic arithmetic operations in bash

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>%</td>
<td>division reminder</td>
</tr>
<tr>
<td>**</td>
<td>exponetiation</td>
</tr>
</tbody>
</table>

Simple integer arithmetics examples:

```
[kkmattil@tunturi ~]$ a=5
[kkmattil@tunturi ~]$ c=3
```
Bash can also use one dimensional array variables, i.e. variables that contain list of items. A specified array item can be recalled by using an index number in brackets with the array variable name (${variable[index]}). For example we can define a simple three element array with command:

array=(a b c)

We can now recall either the whole array or just on element from it. Command:

echo ${array[*]}

own prints out:

a b c

while command:

echo ${array[2]}

prints:

c

Note that in the array, the indexing starts from 0 and thus the sample command above prints out the third element of the array. You can check the number of items in the array by adding # sign to the beginning of variable name. For example in this case command:

echo ${#array[*]}

prints out value:

3

A special case of array variable is $ that holds command line arguments i.e. items that you can provide as input parameters for your script. It the case of this argument array $0 refers to the name of the actual script, $1 refers to the first arguments, $2 to the second and so on. $# refers to the number of arguments and $@ to the full argument list. Below is a sample script that illustrates using the $ array variable:

#!/bin/bash
from_dir=$1
to_dir=$2
mkdir $to_dir
cp $from_dir/*.map $to_dir
If we now execute this script, named e.g. `my_script2.sh` we have to give two arguments for the command. The first argument is in this case used to define a source directory for the copy command and second argument as target directory. For example command:

```
./my_script2.csh  source_data  map_files
```

Would copy all the file with extension .map from a directory named as `source_data` to a new directory called: `map_files`.

### 2.7.3 Quotation marks

Three different quotation marks are used in bash. Quotation marks are frequently needed to define variables and commands to be executed. Following quotation marks can be used:

- `" "` Take text within quotes literally after substituting any variables
- `' '` Take text enclosed within quotes literally
- ``` ``` Take text enclosed within quotes as a command, execute the command and then replace with output of the command to the location of quotation marks

Below are some examples to illustrate the functional differences of different quotation marks. Quotation marks can be used to operate with variables and arguments. When the double or single quotation marks are used all the the text inside the quotation marks are used as one argument. The difference between these two quotation marks is that in when double quotation marks are used, variables are substituted by their values while single quotation marks all text is used as it is. If you run commands:

```
variable=sample1
```

echo "value = $variable"

the result will be

```
value = sample1
```

But if you use single quotation marks instead:

```
echo 'value = $variable'
```

you will get output:

```
value = $variable
```

In linux commands and scripts quotation marks are typically used to define arguments that contain space or other special characters. Say we would like to use `grep` to pick all rows that contain a string `file size` from a file called `files.txt`. Following command **would not work:**

```
grep file size files.txt
```

If you run the command above, you get an error message, as the word `size` is now interpreted to be the second argument defining the input file. We can fix the situation by using quotation marks.

```
grep "file size" files.txt
```

Now the first argument, defining the string to be searched, is `file size`, (including the space between the words) and the second argument, defining the input file, is now `files.txt`, as originally intended.

The third quotation mark type ` ` has a special meaning. With these quotation marks, you can make
one Linux command to produce an argument for another Linux command. The basic syntax ` ` marks is:

```
command1 `command2`
```

where `command1` will use the product of `command2` as an argument. In Bash script, the same functionality can be done also with syntax: `$(command)`

### 2.7.4 Loops and conditional statements

Loops and condition statements are rarely used in interactive command line usage. However they are frequently used in scripts to perform similar commands several times and to control the commands to be executed. Bash provides a wide selection loops, conditional statements and other control structures. In this section we show examples of some of the most commonly used control structures.

A **for loop** performs specified commands iteratively so that on each iteration the loop variable is set to be equal to one of the items in the given element list. In bash a for loop is made with command structure.

```
for variable in element_list
do
  commands
done
```

For example loop

```
for filename in sample1.txt sample2.txt sample3.txt
do
  echo ${filename}
done
```

Would print out

```
sample1.txt
sample2.txt
sample3.txt
```

Typically the argument list contains file names to be processed but it can also be any other parameter too. For example, say we have a directory called `project_3` that contains nine files called: `sample1.txt, sample2.txt ... sample9.txt`. To see the content of the directory we in this case use command `ls`.

```
testuser@hippu1:/wrk/testuser> ls project_3/
sample1.txt sample3.txt sample5.txt sample7.txt sample9.txt
sample2.txt sample4.txt sample6.txt sample8.txt
```

If would like to rename each of these files so that they have extension `.old` we could run command `mv` nine times or we could use a for loop

```
for filename in sample1.txt sample2.txt sample3.txt sample4.txt \
sample5.txt sample6.txt sample7.txt sample8.txt sample9.txt
do
  echo "Renaming file: ${filename}"
  mv project_3/${filename} project_3/${filename}.old
```


done

The for loop above is still quite clumsy as we need to write all the file names to the element list. We can avoid this by substituting the element list with $(ls project_3/). Now, command ls project3 is used to produce a list of file names to be processed:

```bash
for filename in $(ls project_3/)
do
  echo "Moving file: $filename"
  mv project_3/$filename project_3/"$filename".old
done
```

In bash you can also create a for loop where a numerical index variable that is increased automatically by certain step size in each iteration. In this case the syntax is

```bash
for ((variable=start; variable<=end; i++))
```

Below is a for loop that performs the same renaming operation as above, but using just numbers as elements

```bash
for ((number=1; number<=9; number++))
do
  echo "Moving file: sample${number}.txt"
  mv project_3/sample${number}.txt project_3/sample${number}.txt.old
done
```

In while loop, the loop keeps running as long as the defined condition statement is true. In bash a while loop can be made with syntax:

```bash
while [[ condition ]]
do
  commands
done
```

The renaming operation, made above with a for loop could also be done with while loop.

```bash
number=1
while [[ $number -le 9 ]]
do
  echo "Moving file: sample${number}.txt"
  mv project_3/sample${number}.txt project_3/sample${number}.txt.old
  ((number = number + 1))
done
```

In the example above a variable called number is first set to have value 1. The value of this variable is then increased by 1 in the end of each iteration cycle. The iterations are continued until the variable reaches value 10.

Conditional statements (if) can be made as follows.

```bash
if [[ condition ]]
then
  commands
else
  commands
fi
```

You can use operands, listed in table 2.11, in the condition statements of if and while commands. Note, that bash uses different conditional statements for strings and integers. For example the
equality of *strings* is tested with "=" while the equality of *integers* is tested with "-eq". The syntax is also strict about the spaces between the brackets and the condition statement: definition `[[a == b]]` will not work and should be fixed to `[[ a == b ]]`.

**Table 2.11**: Commonly used string, integer and file operands of if and while statements

<table>
<thead>
<tr>
<th>statement</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[[ a == b ]]</code></td>
<td>True if strings a and b are equal</td>
</tr>
<tr>
<td><code>[[ a != b ]]</code></td>
<td>True if strings a and b are not equal to</td>
</tr>
<tr>
<td><code>[[ a =~ b ]]</code></td>
<td>True if strings a and b are similar (allows wildcards)</td>
</tr>
<tr>
<td><code>[[ a &lt; b ]]</code></td>
<td>True if string a is alphabetically before string b</td>
</tr>
<tr>
<td><code>[[ a &gt; b ]]</code></td>
<td>True if string a is alphabetically after string b</td>
</tr>
<tr>
<td><code>[[ a -eq b ]]</code></td>
<td>True if integers a and b are equal</td>
</tr>
<tr>
<td><code>[[ a -ne b ]]</code></td>
<td>True if integers a and b are not equal</td>
</tr>
<tr>
<td><code>[[ a -lt b ]]</code></td>
<td>True if integer a is less than b</td>
</tr>
<tr>
<td><code>[[ a -gt b ]]</code></td>
<td>True if integer a is greater than b</td>
</tr>
<tr>
<td><code>[[ a -le b ]]</code></td>
<td>True if integer a is less or equal to b</td>
</tr>
<tr>
<td><code>[[ a -ge b ]]</code></td>
<td>True if integer a is greater or equal to b</td>
</tr>
<tr>
<td><code>[[ -e name ]]</code></td>
<td>True if file exists</td>
</tr>
<tr>
<td><code>[[ -n a ]]</code></td>
<td>True if string a has non-zero length</td>
</tr>
<tr>
<td>`[[ A</td>
<td></td>
</tr>
<tr>
<td><code>[[ A &amp;&amp; B ]]</code></td>
<td>True if condition A and condition B is true (logical AND)</td>
</tr>
<tr>
<td><code>[[ ! A ]]</code></td>
<td>True if condition A is not true</td>
</tr>
</tbody>
</table>

Below is some examples of *if* command structures.

Check if the integer variable `x` is greater than 10:

```bash
if [[ $x -gt 10 ]]
then
  echo "The value of variable x is more than 10"
fi
```

Check that the variable `x` is greater than 10 but smaller than 20:

```bash
if [[ $x -gt 10 && $x -lt 20 ]]
then
  echo "The value of variable x is more than 10 but less than 20"
else
  echo "The value of x is out of range"
fi
```

You can compare also variables containing text (strings):

```bash
if [[ $answer == "yes" ]]
then
  echo " Your answer was: yes"
elif [[ $answer == "no" ]]
```
then
echo "Your answer was no"
else
  echo "you didn't answer yes or no"
fi

When using less than and more than comparisons you should be careful not to mix string and integer comparisons. For example following condition:

```
[[ 123 > 3 ]]
```

is **FALSE** because string 123 is alphabetically before string 3. The numerical comparison

```
[[ 123 -gt 3 ]]
```

is **TRUE**.

There are a number of operators you can use to test different attributes of a file. The most commonly used operator is `-e` that checks if a file exists. As an example, lets assume that we have a simple list of file names called: `checklist.txt`. Now we would like to check which of these files are found from the current directory. We can use `for` loop to study all the file names and `if` command with `-e` condition to test the file exists.

```bash
for file_name in $(cat checklist.txt)
do
  if [[ -e $file_name ]]
    then
      echo "File $file_name was found"
    else
      echo "File $file_name was not found"
    fi
done
```

2.7.5 **Printing the output**

In the previous examples we have already used `echo` command to write text and variables to the standard output (i.e. to the screen of to a file by standard output redirection). For example command:

```
echo "Hello world"
```

Prints out:

```
Hello world
```

Echo can be used for printing output in many cases, but it does not provide good tools for creating well formatted output with defined columns. In situations where well structured text output is needed, `printf` should be used instead of `echo`. The syntax of `printf` is:

```
printf "format definition" arguments_to_print
```
The format definition defines what types of output is to be printed. Common types include text (%s), integers (%i), and floating point numbers (%f). The format statements can also define how much space is reserved for each argument and how it is located in the column. Below is some simple examples to illustrate the usage of printf command.

Command:

```c
printf "%i %s %s %f\n" 1 Hello World 23.75
```

Prints out:

```
1 Hello World 23.750000
```

Here the format statement defines that the first argument is considered to be an integer, second and third as strings and the fourth argument as a floating point number. Note that by default, printf does not add newline character to the end of the output. To do that the format statement ends with definition \n.

In the next example we define how many characters are reserved for each argument:

Command:

```c
printf "%4i %10s %10s %6.2f\n" 1 Hello World 23.75
```

Prints out:

```
1      Hello      World  23.75
```

Here we reserve four characters for the first integer, then ten characters for each of the strings. The floating point number is presented with six characters, two of which are after the decimal point.

You can also add text and control characters like tabulator (\t) to the format statement. Command:

```c
printf "This is my %i:st %s %s\t %6.1f\n" 1 Hello World 23.75
```

Prints out:

```
This is my 1:st Hello World        23.8
```

In linux scripts printf is typically used to print out values stored in variables. For example commands:

```c
unit=3g
value=5.3
printf "The resulting value from:%4s\t is:\t%f\n" $unit $value
```

Prints out:

```
The resulting value from: 3g is: 5.30
```
3 Directories and data storage at CSC

NOTE: This guide focuses to the common disk environment and directory structure of the supercluster taito.csc.fi and supercomputer sisu.csc.fi. This disk environment is totally independent from the old CSC disk environment that is used in servers, hippu.csc.fi vuori.csc.fi and louhi.csc.fi. The old CSC disk environment is described in the old CSC Data services guide (chapter 4).

The CSC supercomputing environment allows researchers to analyze and manage large datasets. Even with a normal CSC user account the researchers can work with datasets that contain several terabytes of data. Each CSC user has access to four personal disk areas that can be used for processing data. For long term data storage, customers of CSC can use HPC archive server or IDA service.

Knowing the basic features of different disk storage areas is essential, if you wish to use the CSC computing services effectively.

Table 3.1: Standard user directories at CSC. The HPC Archive is used through iRODS commands, and it is not mounted to CSC computing environment as a directory. Directories that are visible to Scientist's User Interface are marked with an asterisk (*).

<table>
<thead>
<tr>
<th>Directory or storage area</th>
<th>Intended use</th>
<th>Default quota/user</th>
<th>Storage time</th>
<th>Backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>$HOME</td>
<td>Initialization scripts, source codes, small data files.</td>
<td>20 GB</td>
<td>Permanent</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Not for running programs or research data.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$USERAPPL</td>
<td>Users' own application software.</td>
<td>20 GB</td>
<td>Permanent</td>
<td>Yes</td>
</tr>
<tr>
<td>$WRKDIR</td>
<td>Temporary data storage.</td>
<td>5 TB</td>
<td>Until further notice.</td>
<td>No</td>
</tr>
<tr>
<td>$TMPDIR</td>
<td>Temporary users' files.</td>
<td></td>
<td>2 days</td>
<td>No</td>
</tr>
<tr>
<td>project</td>
<td>Common storage for project members. A project can consist of one or more user accounts.</td>
<td>On request.</td>
<td>Permanent</td>
<td>No</td>
</tr>
<tr>
<td>HPC Archive</td>
<td>Long term storage.</td>
<td>2 TB</td>
<td>Permanent</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The directories listed in the table above can be accessed through linux command line usage. $HOME and $WRKDIR directories can also be accessed through the Scientist's Interface WWW service. When you are working on command line, you can utilize automatically defined environment variables that contain the directory paths to different disk areas (excluding project disk for which there is no environment variable). So, if you would like to move to your work directory you could do that with command:

cd $WRKDIR

similarly, copying a file data.txt to your work directory could be done with command

cp data.txt $WRKDIR/
In the following chapters you can find more detailed introductions to the usage and features of different user specific disk areas.

![Diagram of user specific disk environment in Sisu and Taito servers.]

**Figure 3.1** User specific disk environment in Sisu and Taito servers.

### 3.1 User specific directories at the servers of CSC

#### 3.1.1 Monitoring disk usage

The amount of data that can be stored to different disk areas is limited either by user specific quotas or by the amount of available free disk space. In Hippu and Vuori you can check your disk usage and quotas with command:

```
quota
```

Below is a sample output for `quota` command:

```
[kkayttaj@taito-login3:~]$ quota
```
The output above shows that the user `kkayttaj` has 50369 files on the `$WRKDIR` directory. These files take 175 GB of the 5120 GB (= 5 TB). In addition the user has 1 GB of data in his home directory.

If the disk quota is exceeded, the user cannot add more data to the directory. The quota can be slightly exceeded for a short time, but after a so called *grace period*, the disk usage must be returned to level that is less than the given quota.

When a disk area fills up, the solution to this is to remove unnecessary files, compress existing files and/or move them to the HPC archive. If you have well-grounded reasons to have more disk space than what your quotas allow, you should send a request to the CSC resource manager (resource_at_csc.fi).

When some of your directories is approaching the quota limits, it is useful to check what files of folders require most of the space. To list the files in your current directory in the order of the file size give command:

```
ls -lSrh
```

Note however, that the command above does not tell how much disk space files that locate in subdirectories require. Thus it is often more useful to use command `du` (disk usage) instead. You can for example try command:

```
du -sh ./*
```

This command returns the size of each file or the total disk usage of each subdirectory in your current directory. You can also combine du command with `sort` to see what file or directory is the largest unit in your current directory:

```
du -s ./* | sort -n
```

Note that as `du` command checks all the files in your current directory, running the command may in some cases take several minutes.

As a best practice, CSC recommends to add read-rights to your files to the members of your group.

### 3.1.2 Home directory, `$HOME`

When you log in to CSC you will first locate in your home directory. The home directory is the storage place for initialization and configuration files and constantly used small programs and files. The size of home directory is rather limited, by default it is only 20 GB, as this directory is not intended for storing and analysing large datasets.

The files stored in the home directory will be preserved as long as the corresponding user ID is valid. This directory is also backed up regularly so that the data can be recovered in the case of disk failures Sisu and Taito servers share the same home directory. Thus if you modify settings files like
.my.cnf, the modifications will affect to usage in both servers.

In linux commands, the home directory can be indicated by the tilde character (~) or by using the environment variable, $HOME. Also command cd without any argument will return the user to his/her home directory.

### 3.1.3 Work directory, $WRKDIR

The work directory is a place where you can store large datasets that are actively used. By default, you can temporarily store up to 5 terabytes of data in it. This user-specific directory is indicated by the environment variable, $WRKDIR. The Taito and Sisu servers share the same $WRKDIR directory.

The $WRKDIR is NOT be intended for long term data storage. Even though un-used files are not automatically removed at the moment (like e.g. in Hippu and Vuori) it is possible that in the future CSC will start using automatic cleaning procedures in this disk area. We will inform Sisu and Taito users, in case automatic cleaning procedures are taken into use.

Further, backup copies are not taken of the contents of the work directory. Thus, if some files are accidentally removed by the user or lost due to physical breaking of the disk, your data is irreversibly lost.

### 3.1.4 Temporary directory, $TMPDIR

Taito server has a directory for storing files during the run. This user-specific directory is indicated by the environment variable, $TMPDIR. It is intended for storing files only during the time when a program is reading and writing them. Each batch job reserves its own temporary directory, and the directory and its contents are purged when the run is completed. The temporary directories of interactive sessions are retained for 24 hours after the last (read/write) action. Backup copies are not taken of the contents of a temporary directory.

### 3.1.5 Software installation directory $USERAPPL

The users of CSC servers are free to install their own application software to the computing servers of CSC. The software to be installed may be developed locally or downloaded from the internet. The main limitation for the software installation is that the user must be able to do the installation without using the root user account. Further, the software must be installed to users own private disk areas instead of the common application directories like /p/appl/bin or /usr/bin. The user application directory $USERAPPL is a directory that is intended for installing users own software tools. This directory is visible to the computing nodes of the server too so software installed there can be used in batch jobs. Unlike the work directory, old files are not removed from this directory. Thus you do not need worry about preserving you software installation. It will stay available for you, until you remove the executables.

The $USERAPPL directories are server specific. This is reasonable: if you wish to use the same software in different machines you normally you need to compile separate versions of the software in each machine. The $USERAPPL directories locate in your home directory and are called: .sisu_appl and .taito_appl. These directories are actually visible to both Sisu and Taito servers. However the in Sisu $USERAPPL variable points to $HOME/sisu_appl, and in Taito to the $HOME/taito_appl.

The default disk quota for $USERAPPL directory is 20 GB.
Below are three examples of using the $USERAPPL directory.

- **Example 1. Installing your own version of program MCL program**
- **Example 2. Installing and using your own Perl module to Taito**
- **Example 3. Installing and using your own Python module at Taito**

**Example 1. Installing your own version of program MCL program**

In this example we install MCL Markov cluster algorithm program to users own $USERAPPL directory Hippu. Installation can be done in the same way in Taito server too.

Once you have logged into Hippu, move to your $USERAPPL directory and create there a new directory called `mcl`.

```
cd $USERAPPL
mkdir mcl
```

Go to the just created `mcl` directory and download the installation package with `wget` command

```
cd mcl
wget http://www.micans.org/mcl/src/mcl-latest.tar.gz
```

In this case the installation package is a **tar-archive** file that has been compressed with `gzip` program. You can unpack this file with commands

```
gunzip mcl-latest.tar.gz
tar xvf mcl-latest.tar
```

After unpacking, the `ls` command shows that a new directory called `mcl-12-068` has been created to your `mcl` directory. This directory contains the actual installation files and documentation of the software. Create a new empty directory called `version-12-064` to the `mcl` directory.

```
ls
mkdir version-12-068
```

After this go to the `mcl-12-068` directory and study its' content

```
cd mcl-12-068/
ls -l
```

Installation packages contain often a short installation instructions. Typically this instruction file is called as **INSTALL** or **README**. In this case you should read the **INSTALL** file to find out how the installation should be done.

```
less INSTALL
```

Many open source software tools are installed using following three steps:

1. Building up the so called **Makefile** with a `/configure` command.
2. Running **make** command that compiles the source code according to the instructions in the **Makefile**.
3. Installing the compiled executables with command **make install**.

Normally the installation packages assume, that the user has permissions to install the software to the locations where the standard linux commands and programs normally get installed. However, at CSC this is not the case. You can install software only to your own disk areas. Often you can use option `--prefix=/path/` to tell to the configure command, where to the program should be installed.
In this case we wish to install the software to the version-09-308 directory in your $USERAPPL area. Thus you must use the following ./configure command:

```
./configure --prefix=$USERAPPL/mcl/version-12-068
```

The configure command checks that all the compilers and libraries, that the software needs, are available. It is not uncommon, that ./configure reports about missing libraries or incorrect compilation options. In those cases you can check if the missing library or program can be taken in use with the module system. In Hippu you can see all the available modules with command module avail. To use certain module, execute command module load module_name. Note also that CSC environment has several compiler versions available. In some cases you may for example need to use certain C-compiler or python version in order to install the software. If you still fail with the installation, ask help from the HelpDesk of CSC.

In the case of mcl the ./configure script runs without error messages, when you use GNU-compilers. The GNU compilers are set up with command:

```
module load PrgEnv-gnu
```

Next need to compile and install the software with commands:

```
make
make install
```

If make and make install commands don't give any error messages, you have successfully installed your software. Typically the executables, i.e. the compiled programs that can be launched, are stored to a sub directory called bin. In this case the bin directory is created to subdirectory $USERAPPL/mcl/version-09-308.

Running command

```
ls $USERAPPL/mcl/version-12-068/bin
```

now shows the programs you have installed:

```
clm clmformat mcl mclcm mclpipeline mcx mcxarray mcxassemble mcxdump mcxi mcxload mcxmap mcxrand mcxsubs
```

The name of the directory that contains the executables may vary between different software. In any case, to be able to use the programs you must tell the location of your own executables to the command shell. This can be done by adding the directory path of you executables to the $PATH environment variable. In this case we add path "$USERAPPL/mcl/version-12-068/bin" to the $PATH variable. This is done with command:

```
export PATH=${PATH}:${USERAPPL}/mcl/version-12-068/bin
```

Note that the first PATH word in the command above is without the dollar sign. Now you can launch the program you have installed. For example

```
mcl -h
```

Remember that also in the future, when you log in to CSC, the PATH variable must be set u before you can use mcl command. Also in the batch job files you need to run the correct export PATH command above before executing the program you have installed yourself.
If the software you have installed works correctly, you can remove the installation package and temporary directories that were used during the compilation. In this case we could remove the `mcl-latest.tar` file and the directory `mcl-09-308/`

```
cd $USERAPPL/mcl
rm mcl-latest.tar
rm -rf mcl-09-308/
```

### Example 2. Installing and using your own Perl module to Taito

The CPAN Perl module archive contains thousands of Perl modules. The default Perl installation contains very limited set of modules while the `Bioperl` installation at CSC contains a bit wider module selection. However, quite often the Perl module that you would need to use is not available at CSC. In these cases you can install the Perl module to your `$USERAPPL` directory. In this example the Perl module `String-Util` is installed.

First, take in use the `perl` version used by `bioperl`. This version is newer than the default `perl` in Taito. It also includes wider module selection.

```
module load bioperl
```

Then go to your `$USERAPPL` and create directory `my_perl`

```
cd $USERAPPL
mkdir my_perl
cd my_perl
```

Download the `String-Util` installation package from the CPAN archive and uncompress and unpack the installation file.

```
wget http://search.cpan.org/CPAN/authors/id/M/MI/MIKO/String-Util-1.21.tar.gz
tar zxfv String-Util-1.21.tar.gz
```

Then move to the String-Util installation directory and compile the package using the standard `perl` installation commands. These installation commands work for most, but not all, perl modules.

```
cd String-Util-1.21/
perl Makefile.PL PREFIX=$USERAPPL/my_perl
make
make test
make install
```

Once you have the `String-Util` module installed, you still need to define `PERL5LIB` environment variable to point to your personal perl module library:

```
export PERL5LIB=${USERAPPL}/my_perl/lib64/perl5/site_perl/5.16.3
```

After defining the environment variable `PERL5LIB` the beginning of your Perl script could look like:

```
#!/appl/bio/bioperl/5.16.3/bin/perl
use String::Util ':all';
```

The installation procedure, described above, needs to be done only once. After the installation you still need to use the `export` command to set the `PERL5LIB` variable, each time you start a new session in Taito.
Example 3. Installing and using your own Python module at Taito

Default Python installation, alike Perl, contains limited set of modules. CSC supports several python versions that can be taken in use with command module load python/version. These python versions normally contain more modules than the default python version.

If Python module of interest is not available at CSC, it can be installed in your $USERAPPL directory. In this example, the Pycluster module (Pycluster.py) is installed. First go to your $USERAPPL and create directory my_python

```bash
cd $USERAPPL
mkdir my_python
cd my_python
```

Download the Pycluster python module installation package from the and uncompress and unpack the installation file.

```bash
wget http://bonsai.hgc.jp/~mdehoon/software/cluster/Pycluster-1.50.tar.gz
tar zxvf Pycluster-1.50.tar.gz
```

Then, move to the installation file and compile the package using the standard python installation command.

```bash
cd Pycluster-1.50/
python setup.py install --prefix=$USERAPPL/my_python
```

In the installation command above, the --prefix option in used instruct the installation script to install the python module to your own python library directory, in stead of the standard library location.

Once you have the Pycluster module installed, you still need to define $PYTHONPATH environment variable to point to your personal Python module library:

```bash
export PYTHONPATH=${USERAPPL}/my_python/lib/python2.7/site-packages:$PYTHONPATH
```

After defining the environment variable PYTHONPATH the beginning of your Python script could look like:

```bash
#!/usr/bin/python
import Pycluster
```

The installation procedure, described above, needs to be done only once. After the installation you still need to use the export command to set the PYTHONPATH variable, each time you start a new session in Taito.

3.2. Archiving data to the HPC archive and IDA storage services

CSC supports two parallel archiving systems for long term data storage:

1. **HPC archive** is intended to storing datasets that are utilized in CSC computing environment.

2. **IDA storage service** is a general storage service for scientific data.

The main difference between between these two services is in their user policy and in the accessibility. The HPC archive is directly bound to the CSC user accounts: All the customers of the CSC computing environment will automatically have an account with 2 TB quota in the HPC
archive.

The IDA service is not directly linked to the CSC computing environment. Even though CSC hosts the IDA service and users need to register to CSC, the storage space is applied from the universities or from the Academy of Finland. IDA users can use the storage space from both their own computers and from the servers of CSC. Thus IDA can be used for transporting data between CSC and local environment. IDA can also be used to publish or share datasets. More information about applying storage space from IDA can be found from the home page of IDA:

- [http://www.csc.fi/sivut/ida](http://www.csc.fi/sivut/ida)

The usage of both HPC archive and IDA is based on iRODS (Integrated Rule-Oriented Data System) technology. Unlike the old $ARCHIVE service, HPC archive and IDA storage space are not directly mounted to the CSC computing environment. The dataset to be stored, is copied to the archive over the internet. The files that are in these storage systems can be managed through iRODS interfaces but the content of the archived files can't be studied or modified. Instead, the stored file must be first retrieved back to the CSC servers or to some other computer in order to analyse or modify the dataset.

Note for Hippu, Vuori and Louhi users:

**The $ARCHIVE service that was previously in use in Hippu, Vuori and Louhi is no longer in use. The files you had in $ARCHIVE have been copied to the HPC archive service.** However, the files form $ARCHIVE are not copied to your HPC archive home directory. Instead they are available in HPC archive directory: `/hpc_archive/old_archive/<username>`.

In CSC environment you can check the content of this iRODS directory with commands:

```
module load irods  ( not needed in Taito)
ils /hpc_archive/old_archive/<username>
```

You can copy a file from this location to your HPC archive home with command:

```
icp /hpc_archive/old_archive/<username>/file /hpc_archive/home/<username>/`

Copying the old $ARCHIVE data to the computing servers can be done with command:

```
iget /hpc_archive/old_archive/<username>/file
```

More information about iRODS and HPC Archive in the main text of this chapter.

### 3.2.1 Using HPC archive

In Taito cluster the iRODS commands are automatically in use. In Sisu, Vuori and Hippu need to run following set-up command in order to be able to execute iRODS commands:

```
module load irods
```
The two basic iRODS commands are:

- **iput** that copies a file to the iRODS server
- **iget** that retrieves a file from the iRODS server

In addition to that, there are several other iRODS commands that can be used to manage the data at the archive server. Many of these i-commands, listed in table 3.1 are analogous to the corresponding linux commands. E.g. Command **irm** removes a file from the iRODS server and **imkdir** creates a new directory to the iRODS server.

We recommend that you don't store all the data to the main folder of the server, but instead you should create a hierarchical directory structure that helps you to locate your files later on. Further, if possible, the files should merged into larger compressed archiving units with programs like **tar** or **zip** before moving data to the HPC archive or IDA.

### Table 3.1 Most commonly used iRODS commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>icd</td>
<td>Change the current working directory (collection).</td>
</tr>
<tr>
<td>ichmod</td>
<td>Change access permissions to collections or data-objects</td>
</tr>
<tr>
<td>icp</td>
<td>copy a data-object (file) or collection (directory) to another.</td>
</tr>
<tr>
<td>iexit</td>
<td>Exit an irods session (un-iinit).</td>
</tr>
<tr>
<td>iget</td>
<td>Get a file from iRODS.</td>
</tr>
<tr>
<td>igetwild.sh</td>
<td>Get one or more files from iRODS using wildcard characters.</td>
</tr>
<tr>
<td>ihelp</td>
<td>Display a synopsis list of the i-commands</td>
</tr>
<tr>
<td>iinit</td>
<td>Initialize a session, so you don't need to retype your password.</td>
</tr>
<tr>
<td>ilocate</td>
<td>Search for data-object(s) OR collections (via a script).</td>
</tr>
<tr>
<td>ils</td>
<td>List collections (directories) and data-objects (files).</td>
</tr>
<tr>
<td>imkdir</td>
<td>Make an irods directory (collection).</td>
</tr>
<tr>
<td>imv</td>
<td>Move/rename an irods data-object (file) or collection (directory).</td>
</tr>
<tr>
<td>ipasswd</td>
<td>Change your irods password.</td>
</tr>
<tr>
<td>iput</td>
<td>Put (store) a file into iRODS.</td>
</tr>
<tr>
<td>ipwd</td>
<td>Print the current working directory (collection) name.</td>
</tr>
<tr>
<td>iquota</td>
<td>Show information on iRODS quotas (if any).</td>
</tr>
<tr>
<td>irm</td>
<td>Remove one or more data-objects or collections.</td>
</tr>
<tr>
<td>irsync</td>
<td>Synchronize collections between a local/irods or irods/irods</td>
</tr>
</tbody>
</table>

### Example 1. Storing data to the HPC archive server

In this example, user kkayttaj copies a set of files from his $WRKDIR directory in Sisu, to his HPC Archive directory.

After logging into Sisu the user sets up the iRODS commands and moves to the work directory of Sisu:

```
kkayttaj@sisu-login5:~> module load irods
kkayttaj@sisu-login5:~> cd $WRKDIR
```
Then the user checks the content of the directory with command \texttt{ls} and creates a new directory called: \texttt{proj27\_data\_1}.

\begin{verbatim}
kkayttaj@sisu-login5:/wrk/kkayttaj> ls
images27\_a.jpg images27\_b.jpg images27\_c.jpg input27.dat
result27\_a.out
result27\_b.out result27\_c.out
\end{verbatim}

kkayttaj@sisu-login5:/wrk/kkayttaj> mkdir proj27\_data\_1

Then the user copies the files he wants to preserve to the new directory:

\begin{verbatim}
kkayttaj@sisu-login5:/wrk/kkayttaj> cp input27.dat proj27\_data\_1
kkayttaj@sisu-login5:/wrk/kkayttaj> cp result27\*\_out proj27\_data\_1
kkayttaj@sisu-login5:/wrk/kkayttaj> cp images27\*\_jpg proj27\_data\_1
\end{verbatim}

After that the user checks that the new directory contains all the files that you wish to store to archive.

\begin{verbatim}
kkayttaj@hsisu-login5:/wrk/kkayttaj> ls proj27\_data\_1
images27\_a.jpg images27\_b.jpg images27\_c.jpg input27.dat
result27\_a.out
result27\_b.out result27\_c.out
\end{verbatim}

Next, the data to be stored is collected to a compressed tar archive file called \texttt{proj27\_data\_1.tgz}.

\begin{verbatim}
kkayttaj@sisu-login5:/wrk/kkayttaj> tar zcvf proj27\_data\_1.tgz proj27\_data\_1
\end{verbatim}

The resulting compressed file \texttt{proj27\_data\_1.tgz} can now be copied to the APC archive. Before copying the data, the user first creates a new sub-folder called \texttt{proj27} to the IDA server.

\begin{verbatim}
kayttaj@sisu-login5:/wrk/kkayttaj> imkdir proj27
\end{verbatim}

Next the user checks that the directory was created to the HPC archive server and changes the current HPC archive server directory as the new \texttt{proj27} directory:

\begin{verbatim}
kayttaj@sisu-login5:/wrk/kkayttaj> ils
/hpc_archive/home/kkayttaj:
   C- /hpc_archive/home/kkayttajl/proj27
\end{verbatim}

kkayttaj@sisu-login5:/wrk/kkayttaj> icd proj27

After this the user is ready to execute \texttt{iput} command that copies the file to to the new directory in the HPC archive server.

\begin{verbatim}
kayttaj@sisu-login5:/wrk/kkayttaj> iput proj27\_archive\_1.tgz
\end{verbatim}

Once the data copying process is finished, the user checks that the file has been successfully copied to the archive:

\begin{verbatim}
kayttaj@sisu-login5:/wrk/kkayttaj> ils -l
/hpc_archive/home/kkayttaj/proj27
   kkayttaj       0 disk-1.4           1344214352  2013-03-25.13:15 & proj27\_data\_1.tgz
\end{verbatim}

After this, the archived files, directory \texttt{proj27\_data\_1} and file \texttt{proj27\_data\_1.tgz} can be removed from the local \$WRKDIR

\begin{verbatim}
kayttaj@sisu-login5:/wrk/kkayttaj> rm proj27\_archive\_1.tgz
kkayttaj@sisu-login5:/wrk/kkayttaj> rm -r proj27\_archive\_1
kkayttaj@sisu-login5/wrk/kkayttaj> rm input27.dat
\end{verbatim}
Example 2. Retrieving data from the archive server

To retrieve the data, stored to HPC Archive in the previous example, the user kkayttaj should do following steps. First the compressed file is copied from the HPC Archive to the $WRKDIR directory.

```
kayttaj@sisu-login5:/wrk/kkayttaj> rm result27*.out
kkayttaj@sisu-login5:/wrk/kkayttaj> rm images27*.jpg
```

```
kayttaj@sisu-login5:/wrk/kkayttaj> module load irods
kkayttaj@sisu-login5:> cd $WRKDIR
kkayttaj@sisu-login5:/wrk/kkayttaj> ils
 /hpc_archive/home/kkayttaj:
 C- /hpc_archive/home/kkayttaj/proj27

kkayttaj@sisu-login5:/wrk/kkayttaj> icd proj27
kkayttaj@sisu-login5:/wrk/kkayttaj> ils
 /hpc_archive/home/kkayttaj/proj27:
 proj27_data_1.tgz

kkayttaj@sisu-login5:/wrk/kkayttaj> iget proj27_data_1.tgz
Then decompress and unpack the data

kkayttaj@sisu-login5:/wrk/kkayttaj> tar zxvf proj27_data_1.tgz
```

After these command the $WRKDIR directory will include directory proj27_data_1 that contains the files stored to the HPC-Archive service.

3.2.2 Configuring the connection to IDA

You can use the IDA storage service directly from the servers of CSC using the same iRODS commands that are used for HPC archive. However, as IDA is not the default iRODS server in the CSC computing environment, you must modify your iRODS settings before connecting to IDA.

Before changing the iRODS setup, close the iRODS connection to the HPC-archive server with command

```
iexit
```

Then make a back-up copy of your iRORS settings file .irodEnv (if you have a one) with command:

```
mv $HOME/.irods/.irodEnv $HOME/.irods/.irodEnv.hpc
```

Then define the parameters for the new iRODS connection with command: iinit. In the case of IDA you should use following definitions:

- iRODS server name: ida.csc.fi
- port: 1247
- iRODS user name: your_IDA_account
- iRODS zone: ida
- iRODS password: your_IDA_password

These iRODS settings are stored into you home directory to files: .irods/.irodEnv and .irods/.irodEnvA. In the case of IDA, running the iinit command is not enough as the location of your IDA home directory needs to be defined in the .irodEnv file. The location of your IDA home
directory depends on your home organization and the name of your IDA project. The syntax of the home directory definition is:

```
irodsHome /ida/your_home_organization/your_project
```

You can add the home directory definition to the `.irods/.irodsEnv` file with a normal text editor or by running a command:

```
echo "irodsHome /ida/your_home_organization/your_project" >> .irods/.irodsEnv
```

Note that the IDA home directory is the same for all the members of your IDA project.

The set-up procedure above need to be done each time you wish to change the iRODS service. (I.e. When you want to shift to from HPC archive to IDA or vice versa).

Once you have configured your iRODS environment, you can start using the storage environment (defined in the `.irodsEnv` file) with the iRODS client commands.

### Example 3. Storing data to the IDA server

In this example, user `kkayttaj` copies a set of files from his `$WRKDIR` directory in Hippu, to his IDA directory. In the example we assume that the user belongs to organization: `jy` (University of Jyväskylä) and IDA project: `jy1234`. The IDA connection has been defined as described in chapter 3.2.2.

After logging in to CSC the user sets up the iRODS commands and moves to the work directory:

```
kkayttaj@hippu4:~>
module load iRODS
kkayttaj@hippu4:~> cd $WRKDIR
```

Then the user checks the content of the directory with command `ls` and creates a new directory called: `proj27_data_1`.

```
kkayttaj@hippu4:/fs/lustre/wrk/kkaytaj> ls
images27_a.jpg images27_b.jpg images27_c.jpg input27.dat
result27_a.out result27_b.out result27_c.out
kkayttaj@hippu4:/fs/lustre/wrk/kkaytaj> mkdir proj27_data_1
```

Then the user copies the files he wants to preserve to the new directory:

```
kkayttaj@hippu4:/fs/lustre/wrk/kkaytaj> cp input27.dat proj27_data_1
kkayttaj@hippu4:/fs/lustre/wrk/kkaytaj> cp result27*.out proj27_data_1
kkayttaj@hippu4:/fs/lustre/wrk/kkaytaj> cp images27*.jpg proj27_data_1
```

After that the user checks that the new directory contains all the files that you wish to store to archive.

```
kkayttaj@hippu4:/fs/lustre/wrk/kkaytaj> ls proj27_data_1
images27_a.jpg images27_b.jpg images27_c.jpg input27.dat
result27_a.out result27_b.out result27_c.out
```
Next, the data to be stored is collected to a compressed tar archive file called proj27_data_1.tgz.

```
kayttaj@hippu4:/fs/lustre/wrk/kkaytaj> tar zcvf proj27_data_1.tgz proj27_data_1
```

The resulting compressed file proj27_data_1.tgz can now be copied to the IDA service. Before copying the data, the user first creates a new sub-folder called proj27 to the IDA server.

```
kayttaj@hippu4:/fs/lustre/wrk/kkaytaj> mkdir proj27
```

Next the user checks that the directory was created to the IDA server and changes the current IDA server directory as the new proj27 directory:

```
kayttaj@hippu4:/fs/lustre/wrk/kkaytaj> ils
/ida/jy/jy1234:
  .apdisk
  ..apdisk
  data_file_1.tgz
  dataset_123.zip
  ..TemporaryItems
  C- /ida/jy/jy1234/proj27
  C- /ida/jy/jy1234/published
  C- /ida/jy/jy1234/.TemporaryItems
```

```
kayttaj@hippu4:/fs/lustre/wrk/kkaytaj> icd proj27
```

After this the user is ready to execute iput command that copies the file to the new directory in the IDA server.

```
kayttaj@hippu4:/fs/lustre/wrk/kkaytaj> iput proj27_archive_1.tgz
```

Once the data copying process is finished, the user checks that the file has been successfully copied to the archive:

```
kayttaj@hippu4:/fs/lustre/wrk/kkaytaj> ils -l
/ida/jy/jy1234/proj27:
  kayttaj 0 disk-1.4 1344214352 2013-03-25 13:15 & proj27_data_1.tgz
```

After this, the archived files directory proj27_data_1 and file proj27_data_1.tgz can be removed from the local $WRKDIR

```
kayttaj@hippu4:/fs/lustre/wrk/kkaytaj> rm proj27_archive_1.tgz
    proj27_data_1
```

```
kayttaj@hippu4:/fs/lustre/wrk/kkaytaj> rm -r proj27_data_1
```

```
kayttaj@hippu4:/fs/lustre/wrk/kkaytaj> rm input27.dat
```

```
kayttaj@hippu4:/fs/lustre/wrk/kkaytaj> rm result27*.out
```

```
kayttaj@hippu4:/fs/lustre/wrk/kkaytaj> rm images27*.jpg
```
4. Running applications at the servers of CSC

4.1 Application software selection at CSC

CSC maintains a wide selection of scientific applications in its servers. The list of officially supported applications can be found from the WWW pages of CSC. A complete list can be from:

http://research.csc.fi/software

You can also find software lists from the science discipline specific pages:

http://research.csc.fi/methods-disciplines

Each of the officially supported software tools have their own instruction page that contains information about where the software is installed, what versions are in use and how the software can be used at CSC (e.g. what module load command sets up the program environment). You should always study the CSC instructions page of the software that you are starting to use at CSC to check the optimal way to use this tool at CSC.

Major part the installed applications are open source tools that can be used without restrictions. Some of the academic programs however require that the user first joins to the user community or accepts software specific usage rules. In the case of commercial applications, like Matlab, Abaqus, Gaussian or Gold, the license usually allows only academic usage and restricts the number of simultaneous users.

In addition to the officially supported software, there is a large amount of other software tools installed to the servers of CSC. Typically these software tools have been installed for testing or for the needs of training courses. You can use these unofficial installations if you like, but you should be aware that they are not actively maintained.

If the application, you would like to use, is not available at CSC you can either install it yourself for your own use or ask CSC to add the tool to the servers of CSC. Normally only software tools, that will have more than just few users, will be added to the selection of software tools supported by CSC. In other cases the CSC staff fill help you to do your own private installation of the tool (see chapter 3.1.5).

Table 4.1 Application software that used most processor time in the servers of CSC on year 2012.

<table>
<thead>
<tr>
<th>Software</th>
<th>Used computing time in CPU years.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gromacs</td>
<td>1695.0</td>
</tr>
<tr>
<td>GPAW</td>
<td>1490.9</td>
</tr>
<tr>
<td>VASP</td>
<td>615.5</td>
</tr>
<tr>
<td>NAMD</td>
<td>378.9</td>
</tr>
<tr>
<td>Gaussian</td>
<td>156.7</td>
</tr>
<tr>
<td>Amber</td>
<td>79.7</td>
</tr>
<tr>
<td>TURBOMOLE</td>
<td>67.4</td>
</tr>
<tr>
<td>Matlab</td>
<td>59.3</td>
</tr>
<tr>
<td>Software</td>
<td>Used computing time in CPU years.</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Octopus</td>
<td>39.4</td>
</tr>
<tr>
<td>BLAST</td>
<td>17.6</td>
</tr>
<tr>
<td>R</td>
<td>10.7</td>
</tr>
<tr>
<td>Molpro</td>
<td>10.2</td>
</tr>
<tr>
<td>NWChem</td>
<td>8.3</td>
</tr>
<tr>
<td>MaterialsStudio</td>
<td>5.7</td>
</tr>
<tr>
<td>Fluent</td>
<td>5.6</td>
</tr>
<tr>
<td>ADF</td>
<td>4.6</td>
</tr>
<tr>
<td>CASTEP</td>
<td>3.7</td>
</tr>
<tr>
<td>Elmer</td>
<td>3.6</td>
</tr>
</tbody>
</table>

### 4.2 Environment module systems

Many, but not all, application software require, that the user loads an environment module before starting the program. The environment module system is needed as the programs and other other tools often have contradictory requirements related to compilers and libraries. For example, it is often necessary to have access to older versions of an application, or to have a library configured with different options for different applications.

The syntax of the environment module loading is:

```sh
module load software_name
```

For example to use Matlab in hippu.csc.fi, you must first run command:

```sh
module load matlab
```

This command adds the Matlab executables to the command path and sets up environment variables that Matlab needs. Only after that you can launch the actual command: `matlab`.

Environment modules provide a convenient way to dynamically change the user's environment so that different compiler suites and application versions can be used more easily. Modules system modifies the environment variables of the user's shell environment so that the correct versions of executables are in the path and linker can find the correct version of needed libraries. For example, the command `matlab` points to different Matlab versions depending of the loaded matlab module.

At the moment CSC is using two slightly different module environment tools:

1. **Environment Modules Project**

Supercomputer sisu.csc.fi, application server hippu.csc.fi and vuori.csc.fi cluster use TCL based module system developed by the Environment Modules Project ([http://modules.sourceforge.net/](http://modules.sourceforge.net/)). In this system all available modules are listed with command `module avail`
Loading module files is done with commands

```
module load modulefilename
```

More information about this module environment can be found from the Sisu user guide.

2. Lmod

Taito.csc.fi supercluster uses a more recently developed version of environment modules called Lmod. It is developed at Texas Advanced Computing Center (TACC) and it is implemented using Lua programming language. In this system, all the installed modules can be listed with command:

```
module spider
```

Lmod uses hierarchical module structure which means that you may first need to load modules higher in the hierarchy before a specific module can be loaded. More details about the Lmod system can be found from the Taito user guide.

4.3 Batch jobs

CSC uses batch job systems to execute computing tasks in clusters and supercomputers (the application sever Hippu does not use a batch job system). Batch job systems are essential for effective usage of large computing servers. First of all, batch job system takes care that the server does not get overloaded: The users can submit large amounts of jobs to be executed and the batch job system takes automatically care that optimal number of submitted jobs are running while rest of the jobs are queueing until sufficient resources are available. Further, most of batch job systems have a "fair share" functionalities that take care that, on the long run, all the users get equal possibilities to use resources. For example in a case where user A has submitted 1000 jobs before user B submits his job, the user B don't have to wait that all the jobs of user A have been processed. In stead, the batch job system gives higher priority to the job of user B compared to user A, as user A is already using much more computing resources that user B.

When batch job system is used, the commands to be executed are not started immediately like in normal interactive usage. In stead the user creates a file that contains the Linux commands to be executed. This command file is typically a Linux script (see chapter 2.7.). In addition to the execution commands this batch job file normally contains information about the resources that the job needs (for example: required computing time, memory and number of cores).

The file is submitted to the batch job system with a job submission command. After that the batch job system checks the resource requirements of the job, sends to job to a suitable queue and starts the job when sufficient resources are available. If the job exceeds the requested results (e.g. requires more computing time than what was requested) the batch job system kills the job. After job submission, user can follow the progress of the job or cancel the job if needed.

In this guide we provide introduction to the SLURM (Simple Linux Utility for Resource Management System) batch job system that is used in Sisu supercomputer, Taito supercluster and Vuori cluster. There are many other batch job schedulers too like Sun Grid Engine (SGE) or Load Sharing Facility (LSF) that are not available in the computer of CSC. The ARC middleware, that is used in the FGI grid, can be considered as a kind of batch job system too. The ARC middleware is available in Hippu application server and in Taito cluster. ARC instructions can be found form the FGI Users' Guide.
4.3.1 Constructing SLURM batch job files

Most common way to use the SLURM batch job system is to first create a batch job file that is submitted to the scheduler with command `sbatch`. You can create batch job files with normal text editors or you can use the Batch Job Wizard Tool, in the Scientist’s User Interface (https://sui.csc.fi/group/sui/batch-job-script-wizard). (See Figure 4.2). In the batch job wizard, you first select the server you want to use and then fill in the settings for the batch job. The batch job wizard can't directly submit the job, but with the "Save Script" you can save the batch job file directly to your home directory at CSC.

Figure 4.2 Batch job wizard in the scientist’s user interface (https://sui.csc.fi/group/sui/batch-job-script-wizard)

Below is an example of a SLURM batch job file made with a text editor:

```
#!/bin/bash -l
#SBATCH -J hello_SLURM
#SBATCH -o output.txt
#SBATCH -e errors.txt
#SBATCH -t 01:20:00
#
 echo "Hello SLURM"
```
The first line of the batch job file ( #!/bin/bash -l) defines that the bash shell will be used. The following four lines contain information for the batch job scheduler. The syntax of the lines is:

```
#SBATCH -sbatch_option argument
```

In the example above we use four `sbatch` options: `-J` that defines a name for the batch job (hello_SLURM in this case) `-o` defines file name for the standard output and `-e` for the standard error. `-t` defines that the maximum duration of the job is in this case 1 hour and 20 minutes. After the batch job definitions comes the commands that will be executed. In this case there is just one command: `echo "Hello SLURM"` that prints text "Hello SLURM" to standard output.

The batch job file above can be submitted to the scheduler with command:

```
sbatch file_name.sh
```

The batch job file above includes only the most essential job definitions. However it is often mandatory or useful to use several other `sbatch` options too. The options needed to run parallel jobs are discussed more in detail in the following chapters. Table 4.2 contains some of the most commonly used `sbatch` options. The full list of `sbatch` options can be listed with command `sbatch -h`

<table>
<thead>
<tr>
<th>Slurm option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--begin=time</td>
<td>Defer job until HH:MM MM/DD/YY.</td>
</tr>
<tr>
<td>-c, --cpus-per-task=ncpus</td>
<td>Number of cpus required per task.</td>
</tr>
<tr>
<td>-d, --dependency=type:jobid</td>
<td>Defer job until condition on jobid is satisfied.</td>
</tr>
<tr>
<td>-e, --error=err</td>
<td>File for batch script's standard error.</td>
</tr>
<tr>
<td>--ntasks-per-node=n</td>
<td>Number of tasks to per node.</td>
</tr>
<tr>
<td>-J, --job-name=jobname</td>
<td>Name of the job.</td>
</tr>
<tr>
<td>--mail-type=type</td>
<td>Notify on state change: BEGIN, END, FAIL or ALL.</td>
</tr>
<tr>
<td>--mail-user=user</td>
<td>Who to send email notification for job state changes.</td>
</tr>
<tr>
<td>-n, --ntasks=ntasks</td>
<td>Number of tasks to run.</td>
</tr>
<tr>
<td>-N, --nodes=N</td>
<td>Number of nodes on which to run.</td>
</tr>
<tr>
<td>-o, --output=out</td>
<td>File for batch script's standard output.</td>
</tr>
<tr>
<td>-t, --time=minutes</td>
<td>Time limit in format hh:mm:ss.</td>
</tr>
<tr>
<td>--mem-per-cpu=MB</td>
<td>Maximum amount of real memory per allocated CPU required by the job in megabytes.</td>
</tr>
<tr>
<td>-p</td>
<td>Specify partitions to be used.</td>
</tr>
</tbody>
</table>

In the second batch job example below options `--mail-type` and `--mail-user` are used to make the batch system to send e-mail to address kkayttaj@uni.fi when to job ends. Further the job is defined to reserve 12GB of memory. In the output and error file definitions `%j` is used to use the job id-number in the file name, so that if the same batch job file is used several times the old output and error files will not get overwritten.
#!/bin/bash -l
#SBATCH -J hello_SLURM
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 01:20:00
#SBATCH -n 1
#SBATCH --mail-type=END
#SBATCH --mail-user=kkayttaj@uni.fi
#SBATCH --mem-per-cpu=12288
#

echo "Hello SLURM"
./my_command

Setting optimal values for the requested computing time, memory and number of cores to be used is not always a simple task. It is often useful to first send short test jobs to get a rough estimate of the computing time and memory requirements of the job. It is safer to reserve more computing time that is needed, but remember that jobs with large computing time request often have to wait longer time in the queue than shorter jobs.

All the batch queues have maximum durations and maximum amount of nodes that a job can use. You can check these limits with command `sinfo -l`. For example:

[kkayttaj@taito ~]$ sinfo -l
Thu Jan 24 08:08:50 2013
PARTITION AVAIL TIMELIMIT JOB SIZE ROOT SHARE GROUPS NODES STATE Nodelist
parallel up 3-00:00:00 1-24 no NO all 257 allocated n[3-11,13-37,39-45,47-110,112-
parallel up 3-00:00:00 1-24 no NO all 257 allocated n[3-11,13-37,39-45,47-110,112-
serial* up 7-00:00:00 1 no NO all 5 idle n[3-11,13-37,39-45,47-110,112-
serial* up 7-00:00:00 1 no NO all 5 idle n[12,38,46,111,127]
longrun up 21-00:00:0 1-12 no NO all 257 allocated n[3-11,13-37,39-45,47-110,112-
longrun up 21-00:00:0 1-12 no NO all 257 allocated n[3-11,13-37,39-45,47-110,112-
interacti up 4:00:00 1 no YES:4 all 2 idle n[1-2]
test up 30:00 1-2 no YES:4 all 2 idle n[1-2]

The `sinfo -l` output above tells that the cluster has five queues (parallel, serial, longrun, interactive and test). The maximum execution time in parallel queue is seven days (7-00:00:00) and the jobs can use use up to 24 nodes (24 * 12 = 288 cores). Similarly the maximum duration of jobs submitted to test queue is 30 minutes (30:00)

Estimating the memory request is even more difficult as it is dependent on several thinks like algorithm and software and the analysis task. In most cases 1-4 GB is enough but you may need to increase the memory size in the case of some application

Command `sjstat` can be used to check the available memory for nodes in different partitions. The `sjstat` command lists the scheduling pool data and the running jobs. The scheduling pool data can be used to check the available memory in different partitions. You can check just the scheduling pool data by adding option `-c` to the command:

`sjstat -c`

Scheduling pool data:

<table>
<thead>
<tr>
<th>Pool</th>
<th>Memory</th>
<th>Cpus</th>
<th>Total Usable</th>
<th>Free</th>
<th>Other Traits</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel*</td>
<td>64000Mb</td>
<td>16</td>
<td>560</td>
<td>558</td>
<td>546</td>
</tr>
<tr>
<td>parallel*</td>
<td>256000Mb</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>serial</td>
<td>64000Mb</td>
<td>16</td>
<td>560</td>
<td>558</td>
<td>546</td>
</tr>
</tbody>
</table>
The sample listing above tells that resource pool "parallel" contains 560 computing nodes, each having 64 GB of memory and 16 cores. In addition to the optimized pool includes also 16 nodes with 256 GB of memory. Of these 556 64 GB memory nodes and 16 256GB memory nodes are free.

Table 4.3 Available batch jobs in supercluster taito.csc.fi.

<table>
<thead>
<tr>
<th>Queue</th>
<th>Number of cores</th>
<th>Maximum run time</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial</td>
<td>12 (one node)</td>
<td>7 days</td>
</tr>
<tr>
<td>parallel</td>
<td>256 (1024 during pilot phase)</td>
<td>3 days</td>
</tr>
<tr>
<td>longrun</td>
<td>12 (one node)</td>
<td>21 days</td>
</tr>
<tr>
<td>test</td>
<td>12 (one node)</td>
<td>30 min</td>
</tr>
<tr>
<td>interactive</td>
<td>12 (one node)</td>
<td>4 hours</td>
</tr>
</tbody>
</table>

4.3.2 Using SLURM commands to execute batch jobs

The basic SLURM commands for submitting batch jobs are `sbatch` that submits jobs to batch job system and `scancel` that can used to stop and remove a queueing running job. The basic syntax of the batch command is:

```
sbatch -options batch_job_file
```

Normally `sbatch` options are included in the batch job file but you can use the options listed in table 4.2 in command line too. For example:

```
sbatch -J test2 -t 00:05:00 batch_job_file.sh
```

If the same option is used both in command line and in the batch job file the value defined in the command line overrides the value in the batch job file. When the job is successfully launched the command prints out the line, telling the ID number of the submitted job. For example:

```
Submitted batch job 6594
```

The job ID number can be used to follow the progress of to job with command `scancel`. For example:

```
scancel 6594
```

The progress of the submitted batch jobs can be followed with commands `squeue`, `sjstat` and `sacct`. These commands can also be used to check the status and parameters of the SLURM environment. Below is some examples for using these commands.

By default `squeue` command lists all the jobs which are submitted to the scheduler. If you want to see status of your own jobs, you can use command:

```
squeue -l -u username
```

or
```squeue -l -u $USER
```

You can also check the status of a specific job by defining the `jobid` with `-j` switch. Using option `-p partition` will display only jobs on that SLURM partition.

Command `scontrol` allows to View SLURM configuration and state. To check when the job waiting in the queue will be executed, the command `scontrol show job jobid` can be used. A row "StartTime=..." gives an estimate on the job start-up time. It may happen that the job execution time can not be approximated, in which case "StartTime= Unknown". The "StartTime" may change, i.e. be shortened, as the time goes.

The `sacct` command can be used to study the log file of the batch job system. Thus it can show information about both active jobs and jobs that have already finished. By default the `sacct` command show information about users own jobs. The `sacct` command has wide selection of options and parameters that can be used so select the data to be displayed. By default `sacct` displays information from the time period that starts from the midnight of current day. You can change the starting date with option `-S YYYY-MM-DD`. For example, to list the information since first of February 2013 you can use command:

```sacct -S 2013-02-01```

Information about specific jobs can be checked with option `-J job-ID`. For example detailed information about job number 6594 could be shown with command:

```sacct -S 2013-02-01 -j 6594 -l```

**Table 4.4** Most frequently used SLURM commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sacct</code></td>
<td>Displays accounting data for all jobs.</td>
</tr>
<tr>
<td><code>salloc</code></td>
<td>Allocate resources for interactive use.</td>
</tr>
<tr>
<td><code>sbatch</code></td>
<td>Submit a job script to a queue.</td>
</tr>
<tr>
<td><code>scancel</code></td>
<td>Signal jobs or job steps that are under the control of SLURM (cancel jobs or job steps).</td>
</tr>
<tr>
<td><code>scontrol</code></td>
<td>View SLURM configuration and state.</td>
</tr>
<tr>
<td><code>sinfo</code></td>
<td>View information about SLURM nodes and partitions.</td>
</tr>
<tr>
<td><code>sjstat</code></td>
<td>Display statistics of jobs under control of SLURM (combines data from sinfo, squeue and scontrol).</td>
</tr>
<tr>
<td><code>smap</code></td>
<td>Graphically view information about SLURM jobs, partitions, and set configurations parameters.</td>
</tr>
<tr>
<td><code>squeue</code></td>
<td>View information about jobs located in the SLURM scheduling queue.</td>
</tr>
<tr>
<td><code>srun</code></td>
<td>Run a parallel job</td>
</tr>
</tbody>
</table>
5. Moving data between CSC and local environment

The disk environment of CSC allows user to deal with very large datasets. However, transferring large datasets to CSC can take a long time. Data transfer performance depends mostly on available bandwidth and distance or latency of the connection. Recent operating systems have automatic TCP buffer tuning feature that improves data transfer performance. Firewalls and VPN-tunnels may cause performance degradation. Transferring a 1 GB file to/from CSC computing environment should usually not take more than a minute or two in an organisation network connected to Funet. Contact IT-support in case of problems.

5.1 Data transport with Scientist's User Interface

Scientist's User Interface (SUI) is a www service that can be used for running applications and managing files in the servers of CSC. You can enter the Scientist's User Interface by using the link "Extranet" that is found in the top of the all normal WWW-pages of CSC (e.g. http://www.csc.fi). The direct address of Scientist's Interface is:


You can log in to the Scientist's User Interface either using your CSC user account and password or by using Haka authentication. If you use the Haka authentication, you must first pick your home organization. After this you can log in by using the user account and password of your own organization (typically Haka uses the windows user name and password of your home institute). Note that even though you would use Haka authentication with your university user name and password, you still need to register as a CSC customer before you can use My Files tool (see chapter 1.2 for registration). Once you have successfully logged in, you can find the tools and functions of Scientist's User Interface from the Services menu in the top of the screen.

5.1.1 Managing files with My Files tool

The file management tools are started by selecting "My Files" from the service list. This selection opens the file management interface, that provides an easy way to transport data between CSC and your local computer. You can also use it to view, edit, compress and remove your files at CSC. Note that some browsers have limitations with file sizes and may not be able to upload larger (more than few gigabytes) files.

The figure 5.1 shows the structure of the file manager. On the MY PLACES field on left side you can see the servers and directories that you can use with the interface. The directory names in the list ($HOME, $WRKDIR) refer to the user directory structure of CSC, explained in chapter 3. Note that some of servers share directories. For example the $HOME and $WRKDIR directories of Taito and Sisu refer to the same directory.
Figure 5.1: The file management tool of Scientist's User Interface.

You can open a directory by double-clicking it. The contents of the directory is shown in the field on the right side. By default the directories are listed alphabetically but you can change ordering criteria by clicking the column names. The files and directories are selected by clicking the corresponding row in the file list.

A pop up command menu opens, when you click the file list with right mouse button. If no no files or directories are selected in the file list, the command menu (figure 5.2.) includes Upload command that you can use to move files from your local computer to CSC.
If you first select a file from the file list, and then press the right mouse button then a different pop up menu, shown in figure 5.3, is show. From this menu you can choose Download command to copy the file to your local computer.

Figure 5.2: The pop up menu of file manager when no files is selected.

Figure 5.3: The pop up menu of file manager when a file is selected.

Note that the list of operations in this menu vary depending on the file type and access permissions. The Open operation sends the content of the file to your browser. This operation is useful when you wish to study e.g. text, image, html and pdf files. Edit operation opens a text file in a simple editor screen. The Pack operation compresses your file or directory using the Zip compression tool. The Properties command opens a window that you can use to check the basic properties of the file and to modify the access permissions of a file.
5.1.2 Transporting large files using the SSH Console tool

In addition to MyFiles Scientist's User Interface provides you another, a bit more complicated tool, that you can use for data transport. That is: using GSI SSH console and SFTP session.

To use this file transport method, Launch the SSH Console by clicking the icon in the Services Desktop or by selecting the tool from the Services Menu. When the console is launched, it asks for your user account and the name of the server you want to connect. The password will be asked only later on. If you would like to copy data to Hippu, you should define:

Username: csc_username
Remote Host: hippu.csc.fi

And then launch the SSH console.

When the Java based console program starts you may need to accept launching the process in your local machine and allow the process connect CSC. When the console is running it will ask you to choose the authentication method to be used. Select: password and press Proceed.

![Figure 5.4: Selecting authentication for SSH Console.](image)

After this a Password Authentication window opens. Type in your password and press OK.

![Figure 5.5: Password authentication for SSH Console.](image)

You can use the console to use the CSC servers, but it contains also other tools than just the command line client. You can use it for example for file transport or secure tunneling of server ports. In the case of file transport you can utilize the SFTP file transport tool. Go to the "Tools" menu of the client window and select:
Tools | SFTP session

Figure 5.6: The tools menu of the SSH console

This launches a file transport terminal to your screen. By default the file transport terminal shows the content of your CSC home directory. To move data to some other location, for example $WRKDIR directory at CSC, change the directory path in the Address filed. Note that you must use the real path of your directory in stead of the environment variable. ( For example: /wrk/username in stead of $WRKDIR ). To upload your file, go to the File menu and select

File | Upload Files

Select the file to be uploaded using the file selection window that opens to your screen and start the upload process. Downloading files is done in the same fashion.
5.2 Copying files from Linux and Mac OSX machines with scp

Copying files between different Linux, MacOSX or UNIX servers can be done with `scp` command. Thus you can use `scp` to transport data between CSC and your local environment or between different file systems at CSC.

The basic syntax for copying data from a local machine to a remote machine is:

```
scp file_to_copy user@remote.server.fi:/path/to/location
```

And correspondingly the syntax to copy files from a remote machine to a local machine is:

```
scp user@remote.server.fi:/path/location/file_name file_name
```

For example user `kkayttaj` could copy the file `data.txt` from his/her local computer to his/her home directory in `taito.csc.fi` server with command:

```
scp data.txt kkayttaj@taito.csc.fi:
```

In the command above, no path or file name is defined in the second argument. This means that the file will be copied to the users home directory with the same name as in the local machine (`data.txt`). If you would like to use other locations or file name, these values can be defined after the `:` sign. For example the following command would copy the file `data.txt` to a directory `data_files` (a sub directory in the CSC home directory) and name the copied file as `data_a2.txt`:

```
scp data.txt kkayttaj@taito.csc.fi:data_files/data_a2.txt
```
To copy complete directories you should use `scp` command with option `-r`. For example the command below copies a directory `data_dir` and all its content to Taito. In this case a directory `data_dir` will be created to directory `/wrk/kkayttaj`. The content of the `data_dir` in Taito will be the same as in the local computer including all the subdirectories.

```
scp -r data_dir kkayttaj@taito.csc.fi:/wrk/kkayttaj/
```

Directory `/wrk/kkayttaj` is the work directory of user `kkayttaj`. Environment variable `$WRKDIR` could also be used to define the location. However, you must enclose the environment variable with quotation marks. The same applies to other directory path defining environment variables too. The following command would copy file `project2.zip` from a local computer directly to the work directory of user `kkayttaj`.

```
scp project2.zip kkayttaj@taito.csc.fi:'$WRKDIR'
```

Copying the data from CSC environment to local computer is done in the same way. If for example file `data.txt` locates in the home directory of user `kkayttaj` it can be copied to the local machine with command:

```
scp kkayttaj@taito.csc.fi:data.txt ./
```

But if the file locates in the work directory it should be copied with command:

```
scp kkayttaj@taito.csc.fi:'$WRKDIR'/data.txt ./
```

You can check the explicit locations of different directories by first logging in to a server in CSC and then giving command

```
echo $VARIABLE_NAME
```

In addition to `scp`, you can also use many other programs for transporting files between local computer and CSC. Many of these tools include graphical user interfaces that make navigation in the file system and selecting files easier. On MacOSX systems you can for example use Fugu file transfer program. ([http://rsug.itd.umich.edu/software/fugu/](http://rsug.itd.umich.edu/software/fugu/))

### 5.3 File transfer in Windows with WinSCP

WinSCP is a free open source program for Windows. It provides a secure file transfer between a local and remote computer. WinSCP supports SFTP, FTP and SCP transfer protocols. WinSCP has many features that make operation with files simple.

#### 5.3.1 Getting started

The installation document for WinSCP is available at the site:

[http://winscp.net/eng/docs/installation](http://winscp.net/eng/docs/installation)

There are two graphical user interfaces in WinSCP: `Explorer` and `Commander`. You select your default interface during the installation of WinSCP, but you can change it later in preferences dialogue. The Explorer interface looks and works similarly as Windows Explorer. If you are familiar with Windows Explorer, you probably had better start with WinSCP Explorer interface.

Start WinSCP and enter your login information like host name, username, password and the server's protocol. In the Explorer interface you can drag and drop files between WinSCP and Windows Explorer to transfer them. In order to do other operations, right-click any object in the interface and select the operation from the pop-up menu. It is also possible to right-click a file or a directory, and
There are many basic operations that you can do with WinSCP:

- Navigating
- Uploading files
- Downloading files
- Managing connections
- Editing/opening files
- Synchronizing local directory with remote one and vice versa
- Changing properties (permissions, ownership, etc.) of remote files
- Renaming files
- Deleting files
- Moving and duplicating remote files
- Creating new objects
- Creating files
- Creating directories
- Creating links and short-cuts

5.3.2 Uploading files to CSC environment

Use can use different methods of WinSCP to upload files to CSC working environment. The easiest methods are drag and drop with a mouse, using copy and paste and using Windows Explorer's "Send To".

When using drag and drop, select your local files you want to upload e.g. inside Windows Explorer. Then drag selected files with mouse and drop them to the remote panel of WinSCP interface.

When using copy and paste, select your local files you want to upload e.g. inside Windows Explorer and copy them to clipboard. Then move to WinSCP interface and select File|Paste from menu (or Ctrl+V). However, if your clipboard content is plain text string only, File|Paste operation works differently. In that case it opens the path stored in clipboard instead of pasting files.

You can use the Windows Explorer's "Send To" feature to upload files to server. To set this feature on, use the installer or select the feature in preferences ("add upload short cut to Explorer's 'sent to' context menu").
5.3.3 Downloading files from CSC environment

The easiest methods for downloading files are drag and drop with a mouse and using URL addresses. When using drag and drop, select remote files you want to download in the remote panel of WinSCP interface. Then drag selected files with mouse and drop them to the local directory e.g. into Windows Explorer.
You can register WinSCP to handle SFTP and SCP protocol URL addresses. To register this feature, use the installer or select the feature in preferences ("Register to handle sftp:// and scp:// addresses"). Then you can type URL in your web browser and WinSCP allows you to download the file.

Figure 5.7. WinSCP session

5.3.4 Further documentation

The extensive WinSCP documentation is available at:

http://winscp.net/eng/docs/start

5.4 Using rsync for data transfer and synchronization

Rsync is a data transport tool that can be used much like the scp command. However, rsync has several features that make it more versatile and in many cases more effective data transport tool than scp. Rsync is a command line program that is available in most Linux and MacOSX machines. In Windows machines you can use rsync through the cygwin system.

The special feature in rsync is that it can transport the data from one site to another in small pieces. During this process rsync first compares the pieces, using checksums, to possibly existing files in the target site, and transports only those pieces that are not found from the target site. In practice this means that if an older or partial version of a file to be copied already exists in the target site, rsync transports only the missing parts of the file. In many cases this makes the data update process
much faster as all the files are not copied each time the source and target site are synchronized.

This feature makes rsync very suitable for maintaining identical copies of a directory in two places (for example between your local computer and a server at CSC). Another situation, where rsync is frequently used is copying large files to and from CSC. Rsync command can be set so that if the data transport process is interrupted, a new rsync based data transport process does not need to start the data transport from the very beginning but it can continue building up the partially transported file.

The Basic command syntax of rsync is:

```
rsync -options source target
```

If the data source or target location is a remote site, it is defined with syntax

```
username@server:/path/in/server
```

However, both the target and source can locate in the same machine too. In that case you can just give directory paths to source and target sites.

The table below lists the most commonly used rsync options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td></td>
<td>Use archive mode: copy files and directories recursively and preserve access permissions and time stamps.</td>
</tr>
<tr>
<td>-v</td>
<td></td>
<td>Verbose mode.</td>
</tr>
<tr>
<td>-z</td>
<td></td>
<td>Compress.</td>
</tr>
<tr>
<td>-e</td>
<td>ssh</td>
<td>Specify the remote shell to use.</td>
</tr>
<tr>
<td>-n</td>
<td></td>
<td>Show what files would be transferred.</td>
</tr>
<tr>
<td>--partial</td>
<td></td>
<td>Keep partially transferred files.</td>
</tr>
<tr>
<td>--progress</td>
<td></td>
<td>Show progress during transfer.</td>
</tr>
<tr>
<td>-P</td>
<td></td>
<td>same as --partial --progress</td>
</tr>
</tbody>
</table>

### Rsync example 1. copying a directory structure from local computer to CSC

Let's assume that user kkayttaj has in his/her local linux or Mac a directory `my_data` that contains files `sample1.txt`, `sample2.txt` and `sample3.txt`. User kkayttaj could use following rsync command to copy the directory to the `WRKDIR` of `taito.csc.fi` server.

```
rsync -avz -e ssh my_data kkayttaj@taito.csc.fi:/wrk/kkayttaj
```

An identical command would be:

```
r sync -avz -e ssh my_data kkayttaj@taito.csc.fi:'WRKDIR'
```

Here the `data source` is the `my_data` directory in local computer and the `target` location is `WRKDIR` directory in `taito.csc.fi`. The option `-a` means `archive mode`. With this selection rsync works recursively which means that all the files and sub-folders from the source folder `my_data` are copied to Taito. The time stamps and access permission of the copied files will be preserved. The `v` option means verbose mode, where rsync prints out information during the transport process. The `z` option defines that the data is transported in a compressed format. The option `-e ssh` defines that ssh
is used to encrypt the data during the transport.

Once the command is executed it asks for the password for Taito and then starts copying the files. If the user now modifies file sample2.txt and runs again the rsync command above, only the sample2.txt file gets updated at Taito. This automatic update process is effective but it includes a very potential danger for accidentally over writing important data! Rsync does not compare the time stamps of the files it will modify. Thus, if the target site contains a newer version of a file than the source site or if the file in the source site is corrupted, then the file in the target site will be replaced by the older or corrupted version when rsync command is executed.

You can test, which files would be affected by the rsync command by using option -n. With this option rsync just lists the target files that differ from the files in the source site, but it does not modify the files in the target site.

rsync -n my_data kkayttaj@taito.csc.fi:'$WRKDIR'

Rsync example 2. Synchronizing directories within a one server

Rsync can be used to synchronize two locations in the same file system. For example say you have a directory called results in both $WRKDIR and $HOME. In the beginning both directories have the same content, but later on new and modified files appear to results directory in $WRKDIR. If you now want to update the changes to the result directory in $HOME too, you can give command:

rsync -avz $WRKDIR/results $HOME/results

Alternative way would be to remove the old results directory from the $HOME and then copy the newer directory to $HOME with commands

rm -rf $HOME/results
cp -r $WRKDIR/results $HOME/

However, that would require copying all the files from $WRKDIR to $METAWRK and thus this process would take much longer than the update done by rsync.

Rsync example 3. Copying a large file from remote site to CSC

Copying files larger than 100 GB over the net can be problematic as often the network connection fails before a copying process, that can take hours, is finished. In these situations rsync is a potential tool as with suitable options, an interrupted copying process can be later on continued with another rsync command.

Now let's assume that user kkayttaj wants to copy to Taito a large file called nz_project.tar.gz from a directory: /project1 at server hpclogin1.canterbury.ac.nz. As the file locates in a server in New Zealand we anticipate that the copying process will take time.

The copying process can be started at Taito with command:

rsync -P kkayttaj@hpclogin1.canterbury.ac.nz:/project1/nz_project.tar.gz $WRKDIR

When the command is executed it first asks for the users password for hpclogin1.canterbury.ac.nz. Note that in this example compression (-z) is not used as the file to be transported is already in compressed format.

In this case option -P, which equals to using options --progress and --partial. With the --progress option, rsync shows how much of the data is already transferred and what is the transport speed.
The --partial selection keeps the partially downloaded files in the target site. This option is useful if the data transfer process gets interrupted by some error or malfunction.

For example, if the rsync command above would terminate before all the data is transported we could launch the same command again:

```
rsync -P kkayttaj@hpclogin1.canterbury.ac.nz:/project1.nz_project.tar.gz $WRKDIR
```

Now rsync would use partial check sums to test the validity of that part of the file that was already transported previously and start the actual data transport only for the missing parts of the nz_project.tar.gz file.

### 5.5 Using wget to download data from web sites to CSC

**Wget** is a handy command for downloading files from the WWW-sites and FTP servers. Once you have resolved the URL of the file, just give it as an argument for wget command to download the file to your current working directory.

```
wget ftp://path/to/the/file
```

For example you could download the nucleotide sequence of human chromosome Y from the ftp-site of UCSC ([ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/](ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/)) with command

```
wget ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chrY.fa.gz
```

The command above would produce a file called chrY.fa.gz to your working directory at CSC (a gzip compressed fasta file). You can also retrieve a group of files by using asterisk (*) sign. For example all human chromosome files could be downloaded with command:

```
wget "ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chr*.fa.gz"
```

Note that quotation marks around the file name are obligatory when asterisk (or any other special character) is used. This command would retrieve all the files, whose name start with chr and end with .fa.gz i.e. the sequence files for all chromosomes.

You can fine tune the behaviour of wget command with several options. You can see the full list of available command options with command:

```
man wget
```

Below is listed some of the most commonly used wget options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i</td>
<td>URL</td>
<td>Read a file containing the URL:s to retrieve.</td>
</tr>
<tr>
<td>-O</td>
<td>file name</td>
<td>Name of the output file.</td>
</tr>
<tr>
<td>-o</td>
<td>file name</td>
<td>Name of the download log file.</td>
</tr>
<tr>
<td>-p</td>
<td>directory</td>
<td>Defines the directory where the downloaded data will be saved to. The default is . (the current directory).</td>
</tr>
<tr>
<td>-c</td>
<td></td>
<td>Continue getting a partially-downloaded file.</td>
</tr>
<tr>
<td>--user=</td>
<td>username</td>
<td>Specify the user name for file retrieval</td>
</tr>
<tr>
<td>Option</td>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>--password=</td>
<td>password</td>
<td>Specify the password for file retrieval.</td>
</tr>
<tr>
<td>-N</td>
<td></td>
<td>Use time-stamping. Download the file only if it is newer that the file in the target directory.</td>
</tr>
<tr>
<td>-m</td>
<td></td>
<td>Turn on options suitable for mirroring. This option turns on recursion and time-stamping, sets infinite recursion depth and keeps FTP directory listings.</td>
</tr>
</tbody>
</table>

For example if you would like to retrieve just chromosomes 2,3,7, you could first collect the addresses of the chromosome file to a single text file:

```
ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chr2.fa.gz
ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chr3.fa.gz
ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chr7.fa.gz
```

If the name of the file is `chr_2.3.7.list` the chromosomes could now be retrieved with command:

```
wget -i chr_2.3.7.list
```

We could also modify the command a bit more. For example command

```
wget -i chr_2.3.7.list -P $WRKDIR -O chr_2.3.7.fa.gz
```

Would retrieve the same files, but instead for producing three separate files, all the files would be concatenated to file `chr_2.3.7.fa.gz` that would be created to the work directory.

### 5.6 Using Funet FileSender to share and transport files

*Fuenet FileSender* is a browser based service for sending large files to colleagues. It provides as an alternative to e-mail attachments but it can be used for moving files to CSC environment too. The *Fuenet FileSender* allows you to upload and send files up to 50 GB size. The service is not intended for long-term storage, but the files are deleted automatically after the preservation time is exceeded. The maximum preservation time is 14 days.

The *Fuenet FileSender* service is available for all Haka-enabled organisations without any additional measures. For example CSC user account is not needed to use this service. Both sending and receiving are possible without installing any additional programs.

You can use *Fuenet FileSender* to share your files with anybody. The receiver doesn't need any authentication for downloading the file from *FileSender*. If you don't belong to Haka, you need a colleague who can send you an upload voucher. The voucher is a one-time permit to use the service for sending a file.

#### 5.6.1 Uploading data to FileSender

First login to the service with your web browser in URL: [https://filesender.funet.fi](https://filesender.funet.fi). You should login to the system with your Haka credentials: first select your home organization and use your local user name and password to log in (not CSC user name and password).

After this you can specify the email address of the recipient and use the **Browse** button to select the file you want to send. After the temporary storage time is defined (**Expiry date:**) and the usage terms are accepted press the **Send** button to upload the file. Once the file is uploaded the recipient will get an e-mail notification.
It is possible to administer your files while they are waiting for the pick-up. Click the *My Files* button to do this.

**Figure 5.8** Funet FileSender file upload page ([https://filesender.funet.fi/](https://filesender.funet.fi/)).

### 5.6.2 Download data from FileSender

The recipient will get an email that contains an URL to the download page of the submitted file. You can start the download by pressing the "Start Download" button. This button is linked to the URL containing the actual file and it can be used to download the file to your local computer.
However, the URL pointing to the uploaded file, can be used also from other applications, not only through the FileSender interface. Thus, if you copy the URL that the Start Download button points to, you can use it with the `wget` command to download the file directly to the CSC servers. The syntax of the `wget` command in this case would be

```
wget "file_sender_url" -O output_file_name
```

For example, if the uploaded file "tophat_test.tgz" has URL:


It can be downloaded to Taito with command:

```
```

### 5.7 Remote disk mounts

With remote disc mounts you can use your CSC directories in a way that resembles the usage of an external disk or USB memory stick. Using this approach normally requires installing some extra software to your local computer, but it also makes the usage very fluent as no `scp` or other data transfer programs are needed to move files between local computer and CSC. Here we introduce `sshfs` tool for creating remote disk mounts in linux machines. A similar remote mounting system, called MacFUSE, exists for MacOSX machines, but it is no longer actively supported.

On linux machines, `sshfs` can be used to mount some disk areas at CSC to users own machine. With this setting the remote disk areas at the servers of CSC can be used just like local directories. To be able to use `sshfs` you local machine must have FUSE ( http://fuse.sourceforge.net/) and sshfs (
http://fuse.sourceforge.net/sshfs.html installed. Once sshfs is installed you can create remote disk mount with command syntax:

```
sshfs username@hostname: mountpoint
```

For example, to make you home directory of user kkayttaj visible to a local computer, execute command

```
mkdir csc_home
sshfs kkayttaj@taito.csc.fi: ./csc_home
```

The first command creates an empty directory that will be used as the mount point in the second command. When the remote mount is established, you can use the directory as any directory in your linux system. For example, to see the content of the CSC home directory of kkayttaj, you could now give a command:

```
ls csc_home
```

By default, the remote directory to be mounted is the users home directory. You can mount other directories too by adding the directory path after the : character. Following commands would mount the $WRKDIR directory user kkayttaj in Taito to a directory taito_wrk on the local machine

```
mkdir taito_wrk
sshfs kkayttaj@taito.csc.fi:/wrk/kkayttaj ./taito_wrkdir
```

Note that the remote directory to be mounted, must be owned by the user. Further, the response times of the remote directories are of cause often much longer than in the case if real local directories.

To unmount the file system, give a command:

```
fusermount -u mountpoint
```

For example the remote mounts, created in the examples above, would be removed with commands:

```
fusermount -u csc_home
fusermount -u taito_wrk
```
# Appendixes

## 1 List of Linux commands discussed in this guide

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apropos</td>
<td>Search commands based on their descriptions.</td>
</tr>
<tr>
<td>awk</td>
<td>Pattern scanning and processing language.</td>
</tr>
<tr>
<td>bg</td>
<td>Move halted process to background.</td>
</tr>
<tr>
<td>bunzip2</td>
<td>Decompress bzip2 compressed file.</td>
</tr>
<tr>
<td>bzip2</td>
<td>File compression tool.</td>
</tr>
<tr>
<td>cat</td>
<td>Print files on the standard output.</td>
</tr>
<tr>
<td>cd</td>
<td>Change directory.</td>
</tr>
<tr>
<td>chmod</td>
<td>Change access permissions for a file or folder.</td>
</tr>
<tr>
<td>cp</td>
<td>Copy a file.</td>
</tr>
<tr>
<td>chprigrp</td>
<td>Change your primary user group.</td>
</tr>
<tr>
<td>cut</td>
<td>Select columns from a file.</td>
</tr>
<tr>
<td>diff</td>
<td>Compare two files.</td>
</tr>
<tr>
<td>du</td>
<td>Show the disk usage.</td>
</tr>
<tr>
<td>echo</td>
<td>Display a line of text.</td>
</tr>
<tr>
<td>emacs</td>
<td>Text editor.</td>
</tr>
<tr>
<td>eog</td>
<td>Image viewing program&quot;</td>
</tr>
<tr>
<td>evince</td>
<td>Viewer for pdf and PostScript files.</td>
</tr>
<tr>
<td>exit</td>
<td>Log out from the system.</td>
</tr>
<tr>
<td>fg</td>
<td>Move halted process to foreground.</td>
</tr>
<tr>
<td>file</td>
<td>Determine file type.</td>
</tr>
<tr>
<td>find</td>
<td>Search for files in a directory hierarchy.</td>
</tr>
<tr>
<td>for</td>
<td>Start a for loop.</td>
</tr>
<tr>
<td>fusermount</td>
<td>Unmount a remote mount created with sshfs.</td>
</tr>
<tr>
<td>gpg</td>
<td>Encryption tool.</td>
</tr>
<tr>
<td>grep</td>
<td>Print lines matching a pattern</td>
</tr>
<tr>
<td>groups</td>
<td>Check which unix group you belong to</td>
</tr>
<tr>
<td>gunzip</td>
<td>Decompress gzip compressed file.</td>
</tr>
<tr>
<td>gzip</td>
<td>File compression tool.</td>
</tr>
<tr>
<td>head</td>
<td>Show the first rows of a file.</td>
</tr>
<tr>
<td>history</td>
<td>Show recet commands</td>
</tr>
<tr>
<td>icd</td>
<td>Change the current iRODS working directory.</td>
</tr>
<tr>
<td>if</td>
<td>Start conditional statement.</td>
</tr>
<tr>
<td>iget</td>
<td>Get a file from iRODS server.</td>
</tr>
<tr>
<td>iinit</td>
<td>Initialize iRODS session.</td>
</tr>
<tr>
<td>ils</td>
<td>List iRODS directories.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><code>imkdir</code></td>
<td>Make an iRODS directory.</td>
</tr>
<tr>
<td><code>iput</code></td>
<td>Put (store) a file into iRODS server.</td>
</tr>
<tr>
<td><code>irm</code></td>
<td>Remove data object from iRODS server.</td>
</tr>
<tr>
<td><code>kill</code></td>
<td>Terminate a running command or process.</td>
</tr>
<tr>
<td><code>less</code></td>
<td>Pager program.</td>
</tr>
<tr>
<td><code>ln</code></td>
<td>Make links between files</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>Show the content of a directory.</td>
</tr>
<tr>
<td><code>man</code></td>
<td>Show the manual page for a command.</td>
</tr>
<tr>
<td><code>md5sum</code></td>
<td>Calculate an md5 checksum for a file.</td>
</tr>
<tr>
<td><code>mkdir</code></td>
<td>Create a new directory.</td>
</tr>
<tr>
<td><code>module</code></td>
<td>Use environment module system.</td>
</tr>
<tr>
<td><code>more</code></td>
<td>Pager program.</td>
</tr>
<tr>
<td><code>mv</code></td>
<td>Move or rename files.</td>
</tr>
<tr>
<td><code>nano</code></td>
<td>Simple text editor.</td>
</tr>
<tr>
<td><code>passwd</code></td>
<td>Change password.</td>
</tr>
<tr>
<td><code>pbunzip2</code></td>
<td>Parallel version of bunzip2 decompression command.</td>
</tr>
<tr>
<td><code>pbzip2</code></td>
<td>Parallel version of the bzip2 file compression tool.</td>
</tr>
<tr>
<td><code>printf</code></td>
<td>Write formatted text to standard output.</td>
</tr>
<tr>
<td><code>ps2pdf</code></td>
<td>Convert a PostScript file to a PDF file.</td>
</tr>
<tr>
<td><code>ps</code></td>
<td>Show current processes.</td>
</tr>
<tr>
<td><code>pwd</code></td>
<td>Show current working directory.</td>
</tr>
<tr>
<td><code>quota</code></td>
<td>Show disk usage in CSC environment.</td>
</tr>
<tr>
<td><code>rmdir</code></td>
<td>Remove a directory.</td>
</tr>
<tr>
<td><code>rm</code></td>
<td>Remove a file.</td>
</tr>
<tr>
<td><code>rsync</code></td>
<td>Tool for data transfer and synchronization.</td>
</tr>
<tr>
<td><code>sacct</code></td>
<td>Show information about executed batch jobs</td>
</tr>
<tr>
<td><code>saldo</code></td>
<td>Tool to check you CPU quota.</td>
</tr>
<tr>
<td><code>sbatch</code></td>
<td>Submit a batch job.</td>
</tr>
<tr>
<td><code>scp</code></td>
<td>Tool for data transfer.</td>
</tr>
<tr>
<td><code>sed</code></td>
<td>Stream editor.</td>
</tr>
<tr>
<td><code>sinfo</code></td>
<td>Show information about the batch job environment.</td>
</tr>
<tr>
<td><code>sort</code></td>
<td>Sort data.</td>
</tr>
<tr>
<td><code>source</code></td>
<td>Execute a command script.</td>
</tr>
<tr>
<td><code>squeue</code></td>
<td>Show the submitted batch jobs.</td>
</tr>
<tr>
<td><code>sshfs</code></td>
<td>Create a remote file mount.</td>
</tr>
<tr>
<td><code>ssh</code></td>
<td>Connect to a remote server.</td>
</tr>
<tr>
<td><code>tail</code></td>
<td>Show the last lines of a file.</td>
</tr>
<tr>
<td><code>tar</code></td>
<td>Data packing tool</td>
</tr>
<tr>
<td><code>tcsh</code></td>
<td>Start tcsh command shell.</td>
</tr>
<tr>
<td><code>tr</code></td>
<td>Translate characters.</td>
</tr>
<tr>
<td><code>uniq</code></td>
<td>Report or omit repeated lines.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>unzip</td>
<td>Decompress zip compressed file.</td>
</tr>
<tr>
<td>wc</td>
<td>Show the number of characters, words and rows for a file.</td>
</tr>
<tr>
<td>wget</td>
<td>Download files from public web sites.</td>
</tr>
<tr>
<td>while</td>
<td>Start while-loop.</td>
</tr>
<tr>
<td>xterm</td>
<td>Open new xterm window.</td>
</tr>
<tr>
<td>zip</td>
<td>Data packing and compression tool.</td>
</tr>
</tbody>
</table>