Scuola Internazionale Superiore di Studi Avanzati

- ma per seguir virtute e conoscenza -

SISSA Unix Manual

SISSA
5th November 2004

Introduction

This manual is written as an introductory guide for unix users at SISSA. It is not meant to be a real unix manual and it is not to be used outside SISSA, since it contains some site-specific information that would probably be of no use on other sites.

Typographical conventions

program	this is a program name
`key`	this is the name of a key you should hit
`something`	this is something you typed and/or is displayed on screen
`something` *different*	as above, plus a parameter you should provide
File → Save as	this is the «Save as» item of the «File» menu

This information is made available as

- a printed manual
- an on-line manual[1] and
- various other electronic formats for off-line reading

About this document

This manual has been last updated on 5th November 2004. It has been written in LaTeX 2_ε and converted to other available formats using standard LaTeX and unix tools and **latex2html**. If you have questions about the manual itself or are interested in the LaTeX source code, please write to `calucci@sissa.it`.

This document has been revised by Viviana Acquaviva, who provided many corrections and helpful suggestions.

[1]http://www.sissa.it/comp/online-manuals/

1 The SISSA Computer System

1.1 Linux

Most computers at SISSA run the Linux[1] operating system. Purists would say that Linux is not UNIX; however most differences are beyond the user level. From a user perspective, if you know Linux, you mostly know Unix too, and vice-versa.

Linux comes in many «flavours» (distributions): we run Red Hat Linux[2] or SUSE Linux[3] on all workstations and most servers.

1.2 Getting a user account

To obtain an account, you are required to fill in the SISSA Account Form, which can be obtained from the Scientific Secretariats or from the Computer Staff desks. The form needs to be signed for approval by the Account Responsible. There is a person in charge of the accounts for each Sector, as in the following table:

	Sector	**Responsible**
AP	Astrophysics	John Miller, Luciano Rezzolla
BP	Biophysics	Vincent Torre, Enrico Cherubini, Stefano Gustincich
CM	Condensed Matter Theory	Stefano De Gironcoli, Stefano Cozzini
CNS	Cognitive Neuroscience	Alessandro Treves, Mathew Diamond
FM	Mathematical Physics	Gregorio Falqui
HE	Elementary Particle Theory	Stefano Bertolini
MA	Functional Analysis and Applications	Gianni Dal Maso, Antonio Ambrosetti
SBP	Statistical and Biological Physics	Paolo Carloni, Cristian Micheletti

Please fill in the form clearly and in capital letters, so as to avoid any misunderstanding and possible errors in the account opening procedure The form duly completed and signed can be handed over to the Computer Staff, room 29 (ground floor), every day before 13.30. The account will normally be operational the same day at 14:30. After 13.30 the account will be operational the next working day.

For visitors and short-term collaborators usually only sector accounts will be opened.

***The use of the account is subject to the laws of the Italian Republic, as well as to standard networking policies*[4].**

Account on the main cluster

The central account provides the following facilities:

- `username@sissa.it` e-mail account
- access to central cluster servers
- access to the ground floor shared workstations

Sector account

The sector account provides the following sector specific facilities:

[1]http://www.linux.org
[2]http://www.redhat.com
[3]http://www.suse.com
[4]http://www.garr.it/docs/garr-aup-00-engl.shtml

- access to the sector server:

sector	server name
AP	`ap-srv.ap.sissa.it`
CM	`cm-srv.cm.sissa.it`
CNS	`hora.cns.sissa.it`
FM	`ubik.fm.sissa.it`
HE	`he-srv.he.sissa.it`
MA	`lagrange.ma.sissa.it`

- access to the sector shared workstations

1.3 Main cluster and sector clusters

The Main Cluster provides essential services to SISSA users (authentication, e-mail, print service, 1TB of storage) and to the external world (web pages). Sector clusters provide sector-specific resources. Please refer to your sector responsible for details.

1.3.1 Shannon and sector gates

The main access point to SISSA is `shannon.sissa.it`. More informations are avilable typing `man shannon` at the prompt.

Several sector-specific gateways are available:

sector	gateway name
AP	`ap-gate`
CM	`cm-gate`
CNS	`cns-gate`
FM	`fm-gate`
HE	`he-gate`
MA	`ma-gate`

1.4 Laptops

Laptops can be connected to SISSA network, provided they have DHCP support enabled.

2 Login and password

2.1 Logging in via ssh

The safest way to log on SISSA computers is by **ssh** (Secure SHell) command. In this way the password you type is encrypted before being sent to the remote computer (in fact, *everything* in a **ssh** session is encrypted). Use of **ssh** command:

`ssh` *machine-name*`.sissa.it`

your-name`'s password:` *(type your password – it is not echoed to screen!)*

The first time you are connecting to a machine, **ssh** will ask you to confirm the connection. Note that **ssh** doesn't ask your username: it takes as default your local username.

You can log in with a different user name typing

`ssh -l` *your-other-name machine-name*`.sissa.it` or

`ssh` *your-other-name*`@`*machine-name*`.sissa.it`

A more advanced use of **ssh** makes use of the so-called *public key authentication*. Once you set up this, you don't need to type any password to log in to remote machines, but only a *passphrase* to unlock your locally-stored *private key*. Note that

- the security of many machines can be compromised if your private key gets stolen, so protect it!
- there is no way to recover your private key if you forget your passphrase: you can not simply ask the staff to enter a new passphrase for you, you need to re-build your keys and set up once again *all* machines to be accessed with them; see also 2.5

Please see **ssh** documentation (`man ssh`, `man ssh-keygen`) for more information.

Public Key Authentication micro-howto

This will tell you how to set up PK authentication in less than three minutes; it will not explain you what PK authentication is. Each step is dangerous. If you don't understand what is going on, please just skip this paragraph.

generate your keys You need first to generate your private and public keys:

```
ssh-keygen -t rsa
```

Be sure to choose a good passphrase: it should be longer and more difficult to guess than your password

copy your public key to remote host Your public key needs to be copied to the remote host:

```
scp ~/.ssh/id_rsa.pub remotehost:.ssh/authorized_keys
```

Be sure you copy the public key `id_rsa.pub`, not the private one (`id_rsa`)! Note that this will overwrite any existing `authorized_keys` file! If you need to append to that file, scp the key to some other temporary file in your home directory, then log in to the remote host and manually append the key to the existing `authorized_key` file

start the agent The **ssh agent** will keep your private key(s) ready for use:

```
eval `ssh-agent`
```

(this fancy syntax is needed because `ssh-agent` outputs some commands that need to be executed by the running shell)

register your private key When started, ssh-agent holds no keys; you need to add them:

`ssh-add -t` *time*

(you will be prompted for your passphrase)

enjoy! Now you should be able to log in to the remote system with the public key only

There is no way to recover a lost passphrase
Always set a lifetime for your keys when using `ssh-add`
Protect your keys

Ssh messages

There are many messages you can receive from **ssh** itself before and during the login procedure. Sometimes you are requested to confirm some action; note that when **ssh** asks you to type «`yes`», it actually means `yes`enter, not just enter or `y`enter.

- `Warning: Permanently added 'storm,192.168.0.2' (RSA) to the list of known hosts.`
  It's the first time you connect to `storm` (from this client); `storm` host key has been automatically added to your database.

-
  ```
  The authenticity of host 'storm (192.168.0.2)' can't be established.
  RSA key fingerprint is 39:16:52:ee:5d:a6:3b:4f:b2:35:d9:8c:7e:9e:b7:1c.
  Are you sure you want to continue connecting (yes/no)?
  ```
  Same as above, but your system setup requires that you accept the key before it is added to the database.

-
  ```
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  @    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
  Someone could be eavesdropping on you right now (man-in-the-middle attack)!
  It is also possible that the RSA host key has just been changed.
  The fingerprint for the RSA key sent by the remote host is
  39:16:52:ee:5d:a6:3b:4f:b2:35:d9:8c:7e:9e:b7:1c.
  Please contact your system administrator.
  Add correct host key in /u/gr/you/.ssh/known_hosts to get rid of this message.
  Offending key in /u/gr/you/.ssh/known_hosts:119
  Password authentication is disabled to avoid man-in-the-middle attacks.
  X11 forwarding is disabled to avoid man-in-the-middle attacks.
  Permission denied (publickey,password,keyboard-interactive).
  ```
  This *can* be bad. However it is also possible that the remote host identification token has been changed for some reason. If you think this is the case, go to your `.ssh/known_hosts` file, remove the *offending key* (in this case located at line 119 of the file) and connect again.[1]

2.2 Logging in via telnet

The use of **telnet** is **strongly discouraged**. **Telnet** does not encrypt what you type at the keyboard nor what comes back to your screen including your username and password. This means that everyone on the Internet may see (and steal) those information. **Telnet** is not supported within SISSA network.

2.3 Logging out

Simply type `exit` at the shell prompt, or press ctrl-d.

2.4 Changing your cluster-wide password

You can change your password at any workstation with the command **yppasswd**:

```
$ yppasswd
Changing NIS account information for your-name on trust.sissa.it.
Please enter old password:   (type your old password – not shown)
```

[1]The `known_hosts` file has very long lines. Be sure not to truncate or split them, or you will break your **ssh** setup.

```
Please enter new password:
Please re-enter new password:
```
(type your new password – not shown) / *(type your new password again)*

Please note that

- the password **is case-sensitive**: «`Password`» is different from «`paSSWorD`»
- the password is changed for the cluster you were logged in only
- some machines *may* cache your password information for a few minutes: this means that the new password *may* appear to be propagated with some little delay

You can use **ypchsh** to change your *login shell*. Please see `man yppasswd` or `man ypchsh` for more information.

In the (unlikely) event you have a local (workstation-only) password, the same rules apply. Only use **passwd** (resp. **chsh**) instead of **yppasswd** (resp. **ypchsh**).

2.5 Choosing a «good» password

When choosing your password you should follow some rules to ensure a reasonable degree of security:

1. **Never** use your first or last name, not even as part of the password
2. **Never** use any word one can find in a dictionary; many password-cracking programs use ten or more dictionaries, so do not rely on your native language being «exotic enough»
3. **Never** use a word one can find in a dictionary with a few digits or special characters prepended or appended: «`19secret!`» is not much safer than simply «`secret`»
4. **Never** use a word one can find in a dictionary with some letter case scrambling: «`pAsswORd`» is not better than «`password`»
5. Use a password length of *at least* six characters (or more)
6. Use a password you can easily remember, e.g. «*M*y *p*assword *i*s *r*eally *a* *g*ood *one*» $\longrightarrow$ «`Mpirag1`» (not *this* one please!)

3 Linux basics

3.1 Graphical environments

When you log in to a workstation you can (and probably will) work in a graphical environment. On most workstations you can choose between the KDE[1] and Gnome[2] (or Bluecurve) environments (possibly others). The first time you log in, you are presented with a brief welcome message, with instructions for getting further help. If you are not familiar with graphical envionments, you are advised to follow the instructions appearing on-screen.

3.2 Text environments

When you log in remotely, or open a terminal emulator on a workstation, you are presented with a text only environment. There are no fancy icons, menus, bars,... here, still there is much work you can do. You will probably find that many jobs are better (and/or easier and/or faster) done in a text environment.

What you interact with when you log in to a text environment is called your *login shell*. From here you have a wide range of programs available. We can roughly divide them as follows:

1. programs that start in the graphical environment – these are actually available/useful only if such graphical environment exists, i.e. they are certainly available from a local terminal emulator on a workstation; they may not be available if you logged in from a remote terminal

   - why should I log in graphically, then open a text environment, then launch a graphical application?
     Short answer: because you like it.
     Longer answer: because you find it simpler to type «`mozilla &`» than to navigate through three levels of menu to start your browser; or because this allows you to start your program with parameters not allowed within the menu interface (e.g. «`nice opera www.your-site.net/ &`»).

2. interactive text-mode programs, e.g. **mc** (file manager), **pine** (mail program), **joe** (editor): when you start such programs, you don't interact with the shell any more, but with the programs you started themselves, until you issue some «quit» command.

3. non-interactive text-mode program, e.g. **ls** (show directory contents), **latex** (text formatter), **finger** (show user information): these programs perform some action and «immediately» return control to the invoking shell

You can get help with most available programs via **man**: `man pine` will show you how to use the **pine** program (`man man` will show the manual for **man** itself). If you know what you want, but don't know what program you could use for it, try `apropos` *keyword*: this will search through all manual page headers for provided *keyword*.[3]

[1] www.kde.org
[2] www.gnome.org
[3] For RedHat 8.0 users: you may need to `export LANG=C` to view correctly **man** pages.

4 Shell basics

4.1 Command line editing and parameter substitution

Modern shells offer fairly advanced command-editing functions. Right and left arrows let you move the cursor withing the command-line you are editing; up and down arrows let you browse your history of previous commands. The `tab` key performs the so-called *expansion* of the word you are currently editing, i.e. the shell looks for a some file name that fits the (incomplete) one you are typing. This is better explained with some examples:

- in the current directory there exists a file named `text-file-1`; you type `pico text-f`tab; the shell completes as `pico text-file-1`
- in the current directory there are two files named `text-file-1` and `text-file-456`; you type `pico text-f`tab; the shell completes as `pico text-file-` and beeps; you press tab two times and the shell shows `text-file-1 text-file-456`, then a new prompt with `pico text-file-` and the cursor at the end of it; you type `4`tab and the shell completes as `pico text-file-456`
- you type `acro`tab; the shell completes as `acroread`, since this file name is the one of an executable file available to you
- in the current directory there exists a file named `somefile.dvi`; you type `xdvi *dvi`tab; the shell completes as `xdvi somefile.dvi` (see 4.1)

The `!last` and `~user` notations

The `!last` notation can be used to repeat a command you already entered to the shell:

- `!!` repeats the last command you entered
- `!abcd` repeats the last command beginning with «`abcd`» you entered
- `!34` repeats the 34th command you entered (you can see the history of your commands with the `history` command)

The `~user` notation can be used to substitute the full path to `user`'s *home directory* (see 4.2): you type `pico ~someuser/somefile` and the shell executes `pico /u/grp1/someuser/somefile`. Your home directory is simply `~/`. Please note that some programs (remarkably **ssh**) treat the tilde as a special character in some situations, so you might need to type it twice to make it actually appear on screen.

Wildcards

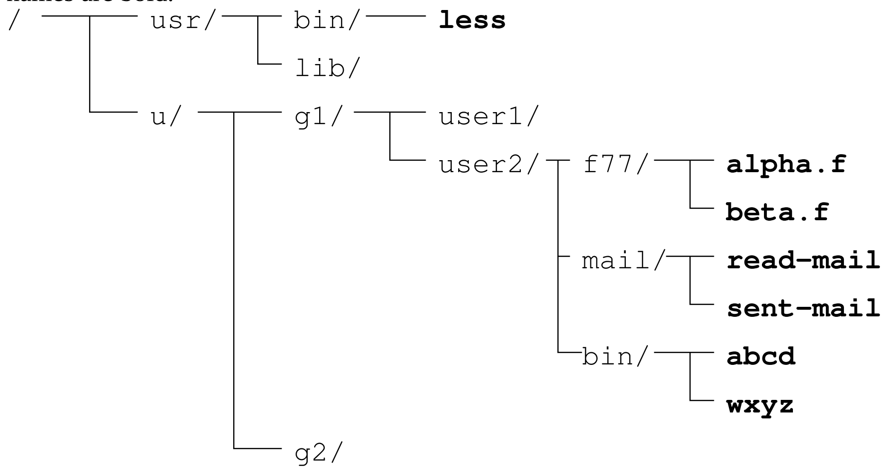
The characters `?` and `*` are special to the shell. `?` means «any character» and `*` means «any number of characters». Thus if you have `file1.tex`, `file2.tex`, `file1.dvi`, `file2.dvi`, typing `ls file?.tex` actually means `ls file1.tex file2.tex`, while `ls file1*` means `ls file1.tex file1.dvi`. Wildcard expansion is performed *before* any command is executed. If you need to pass a `*` or `?` character to some command you need to *escape* it (prepend with a backslash, so `\*` instead of `*`, `\?` instead of `?`) or *single-quote* the word containing it (`'word*'` instead of `word*`, `'word?'` instead of `word?`).

4.2 Working with files and directories

Every file in a unix system can be referred to by means of an *absolute* or *relative path*. The absolute path of a file describes its position referred to the *root* of the whole filesystem; the relative path describes the position of the file as relative to the *current directory*: if the file is in the current directory it can be referred

to with its name only.
This is *not* true for executable files: they are searched for only in certain directories. If you need to execute a file in the current directory you will probably need to write something like `./file-name`

The following picture shows (a little part of) a typical filesystem structure; directory names end with `/`, file names are bold.



If our current directory were `/u/g1/user2/f77/`, we could refer to our fortran sources simply with `alpha.f` and `beta.f`. For any other file we would need to write an absolute or relative path, e.g. a relative path `../mail/read-mail` or an absolute path `/usr/bin/less`. Note that `/` is the *path separator*, and `..` is the parent directory (one level up). Absolute paths always start with the forward-slash `/`.

file & directory commands quick reference

`pwd`	*show current directory name*
`cd`	*change to your home directory*
`cd` *dir-name*	*change to directory* dir-name
`cd ..`	*change to parent directory*
`mkdir` *dir-name*	*create new directory* dir-name
`rmdir` *dir-name*	*remove (empty) directory* dir-name
`cp` *name1 name2*	*copy file* name1 *to* name2
`mv` *name1 name2*	*move (or rename) file* name1 *to* name2
`rm` *name*	*delete file* name
`rm -R` *dir-name*	*recursively delete directory* dir-name *and its contents*
`ln` *name1 name2*	*create a (hard) link to file* name1*, named* name2
`ln -s` *name1 name2*	*create a symbolic link to file* name1*, named* name2
`grep` *string name*	*search* string *in file* name
`find` *path* `-name` `'`*pattern*`'`	*find files whose name matches* pattern*, starting from* path
`diff` *name1 name2*	*find differences between files* name1 *and* name2
`du` *dir-name*	*show space used by directory* dir-name *and its contents*
`df` *dir-name*	*show free and used space in a filesystem*
`quota`	*show your filesystem quota*

Organizing your directories

You should create a logical structure for your files and directories in order to manage them easily. There are several ways of organizing your directory structure:

«flat» structure: all subdirectories put immediately under the top level one; this is probably the simplest structure, but it work well only if the total number of subdirectories is small

«tall» structure: a highly hierarchical structure, useful if you work on many large and complex projects

by project: each project you work on gets its own directory under the top level one; each project directory contains all types of files (and subdirectories) related to the project

by tool: a single directory under the top one for all files created/worked on with a specific tool or programming language; each directory may then contain project subdirectories

by type: a single directory for each file type (executables, images, …)

Mount, mountpoints

For those who come from the Windows world: a unix filesystem is quite different from what you are accustomed to. You do no access «devices» (`A:`, `C:`, …), but a single filesystem structure. Every possible file and directory is found somewhere under the root directory `/`. This does not mean that there is a single storage device: instead, a device can be *mounted* to a directory, i.e. its contents can be made accessible as a sub-tree of the existing filesystem. Mounting and unmounting is usually reserved to the super-user, with the notable exception of removable devices. You can mount the floppy disk with `mount /mnt/floppy`; everything on the floppy will be then accessible from the directory `/mnt/floppy`. Always **remember to unmount** the floppy **before ejecting it**: this is done with `umount /mnt/floppy`.[1] This is needed because the unix filesystem is *asyncronous*, i.e. data are not immediately written to the physical device: if you ejected the floppy before unmounting it, you could lose data. The same `mount`/`umount` procedure is needed for any CD-ROM, which is usually accessed in the `/mnt/cdrom` directory. Most graphical environments provide you with floppy/CDROM icons that automatically perform the mount operation when clicked. Please note that a device cannot be unmounted if any process is currently accessing it, i.e.:

- there are open files
- a directory on the device is the current directory of any process

Links, symbolic links

In a unix filesystem files are not actually *contained* in directories, they are only *linked* from them. As a consequence the following statements hold:

- a file can be linked from 2 or more directories, provided they are on the same filesystem; this means that you can access the *same* file with *different* paths and/or names, so if you see two «identical» files, be sure they are not actually the same file before editing «one» of them, or you will end up with «two» identically edited files
- when you «delete» a file, you only *unlink* it, but the file contents do not disappear if they are linked from some other directory or the file is open (in use by some process); in the latter case the file is actually released as soon as it is closed[2]

Links to existing files are created with the **ln** command; you cannot create hard links to directories. The `ln -s` command is used to create *symbolic links* to files. A symbolic links is a sort of reminder for the filesystem: «when the user tries to access this file name, redirect him to this other location»[3]. Symbolic links can point to directories, and can cross filesystem boundaries; they are not checked before the pointed file is deleted, so you can end up with symbolic links that point to non-existent files.

[1]Please note that the command is `umount`, not `unmount`.

[2]As a consequence, for a device with open but unlinked files *(available space)* $\leq$ *(total space)* $-$ *(space occupied by shown files)*. Some programs actually create large temporary files in `/tmp` and immediately unlink them so you can have little space left on `/tmp` without any files in it!

[3]Unix symbolic links are quite different from Windows `.lnk` files, since they actually contain nothing but their name (in fact they do not occupy any space on device) and no special software is required to perform the redirection (instead, special care is required to read the link itself)

M- commands

If you ever worked with *MS-DOS*, you may feel more comfortable using the tools provided by the *mtools* package. Most useful tools are **mcd**, **mcopy**, **mdel**, **mdir**, **mformat**, **mmd**, **mmove**, **mrd**, **mrem**, **mtype** that perform the same actions as the corresponding *MS-DOS* commands, and can be used when working on files on a FAT filesystem (e.g. most floppies). Syntax and behaviour are the same as *MS-DOS* commands, *when applicable*. This means that:

1. you can use the `A:` notation to refer to the floppy disk (if present on the system, a Zip drive is attached to `Z:`, and a Jaz drive to `J:`)
2. file names are displayed and referred to in the `8+3` format
3. file names are case insensitive
4. you do not need to mount and umount any media to access files on it
5. the path separator is the backslash \, not the forward slash /

   but

6. you still need to use unix syntax when copying to or from a unix filesystem

Mcopy will also handle character encoding and carriage return conversion if instructed to do so (`-a` and `-T` switches). See `man mtools` for a complete list of available **mtools**.

4.3 Redirection and pipelines

Many unix command read their input (if any) from the so-called *standard input* (`stdin`) and write their output to the so-called *standard output* (`stdout`), plus optional warnings to the so-called *standard error* (`stderr`). When launching them from an interactive shell, `stdin` is normally connected to the keyboard, while `stdout` and `stderr` are both connected to the terminal screen.

Sometimes you will find it useful to *redirect* standard input, output and/or error, i.e. make the running command read from or write to some file instead of keyboard and screen. The < operator is used to redirect `stdin`; the > operator is used to redirect `stdout`; the `2>` operator is used to redirect `stderr`; the `>&` operator is used to redirect both `stdout` and `stderr` to the same place.

- `ls > somefile` will print the current directory contents to `somefile`
- `nc somehost 10999 <somefile` will netcat `somefile` to port 10999 of «somehost»

When redirecting output, you can append to an existing file instead of overwriting it; just substitute > with ». Mind that the behaviour of the > and » operators with existing (resp. non-existing) files is shell- and setup-specific; refer to shell manuals for details.

Recent versions of the **bash** shell can redirect output to a remote host via tcp or udp with the following notation:

(tcp) `somecommand >/dev/tcp/`*host name*/*port number*

(udp) `somecommand >/dev/udp/`*host name*/*port number*

(note that `/dev/tcp/` and `/dev/udp/` are not actual directories)

You can even redirect the output of one command to be the input of another command: this is done with the `|` (*pipe*) operator. It is quite common to have several commands connected by redirection operators in a *pipeline*:

```
cat mybadPS.ps | fixps | psbook -q | psnup -2 > mybook.ps
```

In this pipeline, `fixps`, `psbook` and `psnup` are said *filters*, i.e. data flow through them without hitting the disk.

4.4 Searching for files

The **find** utility can search a directory (sub-)tree for files matching a pattern; **find** is invoked as

```
find directories options
```

where `directories` is a list of directories to search (often `.` only) and `options` is a list of options to select files (and possibly do something with them).

some find options

`-name 'pattern'`	find files with names matching the specified pattern
`-iname 'pattern'`	find files with names matching the specified pattern, case insensitive
`-mtime -days`	find files modified less than `days` ago
`-mtime +days`	find files modified more than `days` ago
`-mmin -minutes`	find files modified less than `minutes` ago
`-maxdepth n`	do not descend more than `n` levels of subdirectories
`-type f`	find regular files only
`-type d`	find directories only

Example: find C source files in my home modified in the last 5 days

```
find   -mtime -5 -name '*.c'
```

There are *many* more options for **find**, but some of them are quite tricky, so please be sure to understand the full man page before experimenting with them.

Please note that `find /usr/bin -name some-program-name` is **not** the right way to find the location of an executable: `which some-program-name` is shorter to type and faster (and will also search directories other than `/usr/bin`). If you are looking for a «system» file (i.e. a file located in `/usr` or `/etc`) use `locate file-name`; `locate` uses a database of files that is reindexed every night, so it is *very* fast, but its output does not reflect last minute changes.

4.5 Searching for strings

The **grep** utility can be used to search for strings or patterns; **grep** can work on regular files or as a *filter* (see 4.3):
`grep somestring somefile` will output every line of `somefile` containing `somestring`;
`cat somefile | grep somestring` will do exactly the same.

some some options

`-C n`	show `n` lines of context, i.e. show `n` lines before and after the matched line
`-c`	suppress normal output, show only the number of matching lines
`-i`	perform case-insensitive match
`-l`	suppress normal output, show only the names of files with matching lines
`-n`	show line numbers
`-r`	descend recursively into subdirectories
`-v`	show *non*-metching lines

Grep can search for simple strings or for more complex patterns, also known as *regular expressions* (*regex*). Regular expressions are strings where some characters have special meanings:

some grep regex components

`.`	any character
`\?`	multiplier: 0 or 1 time
`\*`	multiplier: 0 or more times
`\+`	multiplier: 1 or more times
`|`	logical or
`\(\)`	grouping

grep regex examples

`Password|password`	matches any line containing «password» or «Password»
`Aa\*rgh!\+`	matches «Argh!», «Aaaargh!», «Aaargh!!!», ...

4.6 Shell scripting

Much more work can be done using shell *scripts*, i.e. (usually) short «programs» written in the language of the shell: one can use variables (maybe arrays too), conditionals, loops, pattern matching and substitution, simple arithmetics and more. Please refer to shell manuals for this.

Some example scripts

Use at your own risk: do not complain if anyone of these scripts deletes all your files, locks your account or kills your cat.

Countdown

```
#!/bin/bash

if [ $# == 0 ]; then          if zero command-line arguments
    timeout=10
else
    timeout=$1          the 1st command-line argument is the desired timeout
fi

printf "%3d" $timeout
while :  ; do          infinite loop
    sleep 1
    timeout=$((timeout-1))
    printf "\b\b\b"
    printf "%3d" $timeout
    if [ $timeout == 0 ]; then
        printf "...expired !\n\a"
        exit 0          jump out
    fi
done
```

This is not actually very useful, but shows plenty of **bash** features: command-line arguments handling (`$1`, `$#`), conditionals (`if ...else ...fi`), loop (`while ...do ...done`).

Open ssh session in a new xterm

```
#!/bin/sh

xterm -T $1 -e ssh -t $* &
```

Much shorter, a bit more useful.

Mass conversion

```
#!/bin/sh

for im in *.gif ; do
    name=`basename $im .gif`
    convert $im ${name}.png
done
```

Convert all GIFs (in the current directory) to PNGs, using ImageMagick `convert` tool.

Environment and shell setup

Various scripts are automatically executed by the shell at startup and shutdown. The following table shows which files are executed in various situations by **bash** and **tcsh**. Files in `/etc/` are owned by the super-user and are read-only, as far as you are concerned. Files in your home directory (`~/`) can be customized; they are all «hidden» files (name begins with a dot), so they do not show up with `ls`; use `ls -a` to list all files, including hidden ones, or `ls -d .*` to list hidden files only.

bash	**tcsh**
at login	

`/etc/profile`
`~/.profile`
`~/.bash_profile`
`~/.bash_login`
	`/etc/csh.cshrc`
	`/etc/csh/login`
	`~/.tcshrc` *or* `~/.cshrc`
	`~/.login`
	`~/.cshdirs`

at startup of a non-login interactive shell	

`~/.bashrc`	`/etc/csh.cshrc`
	`~/.tcshrc` *or* `~/.cshrc`

at logout	
`~/.bash_logout`	`/etc/csh.logout` `~/.logout`

There is no conceptual limit to what you can do from these scripts. However they usually set up some *enviroment variables*, i.e. little pieces of information that can be read from subsequently starting programs. To create an environment variable you use the `export VARIABLE=value` syntax when working with **bash** and `setenv VARIABLE value` when working with **tcsh**. To read the value of a variable use `$`*`variable_name`*, e.g. `echo $HOME`.

common environment variables

`$PATH`	*colon-delimited list of directories where to look for executables*
`$HOME`	*your home directory*
`$PWD`	*shell current directory*
`$MAIL`	*your INBOX folder*
`$LANG`	*this is related to character encoding – RedHat 8.0 users should probably set* `LANG=C`
`$DISPLAY`	*where to display graphics windows – if you logged in via* **ssh** *this has already been set for you*
`$EDITOR`	*which program to start when an editor is needed by some other command*
`$?`	*status of the most recently executed foreground command (0 means success)*
`$$`	*process ID of the running shell*
`$!`	*process ID of the most recently executed background command*

5 Editors

There is a large number of text editors available for unix systems. For historical reasons «the» editors are **vi** and **emacs**. There is at least one good reason to learn the basics of vi and emacs: they are available almost everywhere. In fact many unix commands use **vi** as default editor.

5.1 vi

Start the **vi** editor with `vi` *your file name*. The **vi** editor has two running modes: command and insert mode. Command mode is the default at startup. In this mode all letters on your keyboard have a meaning and send a command to the editor. To switch to insert mode and insert a text into the file you must press `i` (or `insert`) or `a` (add). After that everything you type in your keyboard will be inserted in the file as text. To exit the insert mode and return to command mode hit `esc` or `ctrl-[`.

vi command quick reference

cursor movement	arrow keys
scroll up	`ctrl-u` or `ctrl-b`
scroll down	`ctrl-d` or `ctrl-f`
go to top	`1` `G`
go to bottom	`G`
go to line number *n*	n `G`
find backward	`?` *string to search* `return`
find forward	`/` *string to search* `return`
find next (same direction)	`n`
find next (opposite direction)	`N`
delete character at cursor	`x`
delete a line	`d` `d`
delete to end-of-line	`d` `$`
delete word at cursor	`d` `w`

5.2 emacs

When you are using **emacs** the screen is divided into a number of windows. The text window can be divided horizontally or vertically into multiple text windows, each of which can be used for a different file. The complete manual of **emacs** is available in the S.I.S.S.A. library. There is nothing special you have to do to write your text, all commands are issued using the `ctrl` key.

emacs command quick reference

save buffers and exit	`ctrl-x` `ctrl-c`
delete character at cursor	`del`
move to the beginning of the line	`ctrl-a`
move to the end of the line	`ctrl-e`
refresh screen	`ctrl-l`
delete to the end of line	`ctrl-k`
display help	`ctrl-h`
find forward	`ctrl-s` *string to search* `return`
find backward	`ctrl-r` *string to search* `return`

5.3 Other popular editors: pico, mcedit, joe

pico is a very simple, display-oriented text editor. Commands are displayed at the bottom of the screen, and context-sensitive help is provided. As characters are typed they are immediately inserted into the text. Editing commands are entered using the `ctrl` key. See `man pico` or the internal help, accessed with `ctrl-h` from **pico**.

mcedit is the internal editor of the **mc** (Midnight Commander) file manager. All commands are available via hot-keys `F1` to `F10` as shown at the bottom of the screen, or via the menu system.

joe is another powerful editor. Key-bindings for useful commands are shown on-screen hitting `ctrl-k` `h`. **joe** can mimic the behaviour of other editors if invoked with different names: **jstar** (Borland Wordstar), **jmacs** (emacs), **jpico** (pico).

5.4 X11 editors

Many other editors are available when working in graphical environments: **xemacs** (X11 version of emacs), **kate** and **kwrite** (KDE environment), **gedit** (Gnome environment), **nedit**. They all have a menu-driven interface and an on-line help.

6 Using printers

6.1 Printing and managing print queues

Several network printers are available at SISSA:

printer name	**location**	
`pst`	ground floor	
`ps12`	1st floor	
`pscc`	1st floor	*color printer*
`ps22`	2nd floor	
`psc`	2nd floor	*color printer*
`psnb1`	«new» building	
`psnb2`	«new» building	
`psnb3`	«new» building	

Every workstation has a default printer, which is usually the closest one. When printing from some graphical application you have usually to find a `Print` command, probably available inside the `File` menu.

You can use the **lpr** command to print from the command line (in fact, most «`File`⟶`Print`» interfaces just put somewhere a temporary file and invoke lpr onto it). Please be sure to feed lpr with text or Postscript files **only**. Text files can be converted to postscript with **a2ps**. You can print to a specific printer with the `-P` switch, e.g. `lpr -Ppsc somefile.ps` will print `somefile.ps` on the color printer at 2nd floor. Please use color printers for color jobs only.

Once you issued the `lpr` or `File`⟶`Print` command, your job is sent to the *print spooler* and possibly queued after other jobs previously sent to the same printer by other users (or by yourself). You can then examine the *print queue* to see if your job gets printed or to remove it. This can be done with the **lpq** command (with the same `-P`*printer* switch you used for `lpr`) or via the web interface[1]. Command-line removal of a job in the printing queue is done with the **lprm** command.

`lpr -Pps12 somefile.ps`	*send* `somefile.ps` *to 1st floor printer*
`lpq -Pps12`	*show 1st floor printer queue*
`Printer: ps12@spooler ...`	
`...`	*(this is the output of* `lpq`*)*
` Rank Owner/ID ...`	
`1` *your name*`@`*some host*`+951 ...`	*your job has been assigned number 951*
`lprm -Pps12 951`	*remove it*

Please note that **all printers** at SISSA use ***A4* paper**, and can occasionally get stuck if a job that uses a different paper size (e.g. *letter*) is submitted.

Many more printing options are available using the `xpp` command on `shannon`.

a2ps

a2ps is a quite complex program, despite the fact it is mostly used only to convert plain text to (possibly multi-column) postscript. It provides many other features, such as syntax highlighting and specialized processing for many common text formats. There is a small manual available with `a2ps --help` or `man a2ps` and a *much* larger one with `info a2ps`. System settings for **a2ps** are defined in `/etc/a2ps.cfg` and `/etc/a2ps-site.cfg`. Most of them can be customized by the user, writing appropriate definitions in the file `~/.a2ps/a2psrc`.

6.2 Printing policy

Some printing guidelines:

[1]http://spooler.sissa.it/

1. do not use color printers for black-and-white only jobs
2. if you need many copies of the same document, please use photocopiers instead of printers
3. whenever appropriate use duplex printing[2]
4. whenever appropriate use **a2ps** to convert plain text to postscript: it will produce a more compact output, and will add some useful syntax highlighting to many programming languages
5. if your job doesn't get printed, check the print queue before re-submitting it: it may have produced some error to the printer, or it may have been queued after some other job
6. if the printer becomes «confused» and starts printing hundreds of pages of junk, interrupt the job – this usually happens when the printer is fed with input of unknown format and can result in a thousand sheets of paper wasted and the print queue blocked for hours

In order to prevent abuses, every print job is recorded in a central database as soon as it is printed. Recorded data are *not* available to users, please do not ask for them.

[2]this is the default for all black and white printers

7 Processes

Whenever you start a program from the shell, a new process is created. Sometimes you may want to control how the process is running, and several commands are available for this task.

7.1 Background and foreground

You usually start commands in the *foreground*, i.e. in such a way that what you type at the keyboard is directly sent to the running process and its output comes back to you (but see 4.3). While this is strictly necessary for some programs you need to interact with (e.g. **pine**), sometimes you could prefer to start a process in the *background*, so that you immediatly get back the shell prompt (e.g. you compile a really big project – you will probably redirect output to some file). This is done appending the `&` operator to the command line: `make all >& /tmp/make-all.output &`.[1] You get something like `[3] 342` and a new command prompt: `[3]` is the *job identifier*, and can be used to control the job *within the launching shell only*; `342` is the *process identifier* (PID) and can be used system-wide. Once the make is done you will get `[3]+ Done make all >& /tmp/make-all.output` just before a command prompt.

job control quick reference

`command`	*start* `command` *in the foreground*
`command &`	*start* `command` *in the background*
`ctrl-c`	*kill current foreground job*
`ctrl-z`	*stop current foreground job*
`fg`	*continue in the foreground last stopped job*
`fg %`*job-id*	*continue in the foreground job* job-id
`bg`	*continue in the background last stopped job*
`bg %`*job-id*	*continue in the background job* job-id
`kill %`*job-id*	*kill running job* job-id
`kill` *pid*	*kill running process* pid *(this can be used from other shells)*
`killall` *name*	*kill running process(es) named* name
`ctrl-s`	*suspend terminal output*
`ctrl-q`	*resume terminal output*

7.2 Nice

Concurrent processes are scheduled to run based on their own priority. You can alter the way your processes are scheduled with the **nice** and **renice** commands. You (the user) are only allowed to *lower* the priority of processes – only the super-user can do the opposite. Low-priority processes will get less CPU time on a busy system, thus allowing other processes to run faster; still they will run at full speed if no other process is in the run queue.

nice quick reference

`command`	*start* `command` *at normal priority*
`nice command`	*start* `command` *at low priority*
`renice -p` *pid*	*lower priority of process* pid

[1] Analysis of this command (pay attention to the different meanings of `&`):

`make all`	start `make` with parameter `all`
`>&`	redirect both `stdout` and `stderr`…
`/tmp/make-all.output`	…to this file
`&`	and go to background

7.3 Signals

Running processes can be controlled using *signals*. Signals are sent using the **kill** command: `kill -`*signal-number pid* or `kill -`*signal-name pid* will both send a signal to process *pid*. The list of available signals is shown with `kill -l`.

common signals

STOP	*stops a running process – actually* `ctrl-z` *sends this signal*
CONT	*continue a stopped process*
INT	*kill a process – actually* `ctrl-c` *sends this signal*
TERM	*kill a process*
KILL	*kill a process*
signal KILL can not be blocked, i.e. the process will surely die	

7.4 Diagnostics

A few commands are available to inspect process status.

ps will show almost any possible kind of information of any running process. It has quite a lot of command line options, so please see `man ps`.

top is an interactive process viewer. Its default configuration will show the «most active» processes running on the system, but this can be changed to show processes from one user only or processes with the largest memory usage. It can be used to send signals or renice processes too. An on-line help is available hitting `h`. Various top-like programs are available in graphic environments (gtop, SystemMonitor, . . .); note that these programs need a large amount of resources just to redraw their screen, so their reports may be not so accurate (e.g. SystemMonitor will use from 5 to 10% of CPU time on a P4/2.4GHz).

uptime will show the time since last reboot, the number of users logged in and three *load average* numbers: these are a moving mean of the system activity during the past 1, 5 and 15 minutes. If you see large values, don't be surprised if the system slows down.

limit will display memory size and CPU time limitation for user processes. This is an internal command of the **tcsh** shell; the corrisponding for **bash** is `ulimit -a`. CPU time limits for user processes are enforced on remote accessible, general purpose machines (currently `shannon`).

8 E-Mail

When an account at SISSA is opened for you, you are given an e-mail address *your-name*`@sissa.it`. Some sectors may provide you with a sector e-mail address like *your-name*`@`*sector*`.sissa.it`. SMTP and POP services are provided by `mail.sissa.it`, IMAP service is provided by `imap.sissa.it`.

8.1 Reading and sending mail in text mode

The suggested mail program for text mode is **pine**. It features an intuitive interface, with command help always available on-screen. While running pine will check for new incoming mail at regular intervals. A large help file is provided hitting the ? key at the startup screen of pine. **Pine** folders are compatible with webmail. **Pine** will read HTML mail and run external «helpers» to display attachments.

8.2 Reading and sending mail with X

Several mail programs are available for graphical environments. The **Netscape Communicator** (or **Mozilla**) browser has an internal mailer; **KMail** is the KDE mail program, while recent Gnome installations provide **Ximian Evolution**. All of these have on-line help available.

8.3 Webmail

A webmail interface is available at `https://webmail.sissa.it`. This interface allows you to access your mailbox from anywhere in the world with a web browser, presenting a consistent view of all your mail folders (the folder structure managed by the webmail system is compatible with the one generated by **pine**). The webmail interface already allows you to set up **mail filters** for incoming mail and to manage your **address book**, with import and export of contact list from and to popular formats. We are in the process of extending the webmail interface with a **calendar**, a **task list**/**todo manager** and **mail auto-reply** and **forwarding**. For these reasons webmail will be the **suggested** way to read and write mail at SISSA.

9 WWW and FTP

9.1 Available browsers

On most machine the default browser is **Netscape Navigator** or **Mozilla**. However on many workstations you can choose also **Galeon** (Gnome), **Konqueror** (KDE) or **lynx** (text-only browser). There are no special issues with any of this browsers. Please note only that when clicking on a `mailto:` URL each of them will start a different mail program (the internal mailer for Netscape/Mozilla, Evolution for Galeon, KMail for Konqueror) so you risk to end up with messed mail folders if you use too many of them.

The web without browsers

Sometimes it can be useful to download information from the web without having to interact with a browser: maybe you need to download a large number of related small pages, or you find the web server is terribly slow, except from 3 to 5AM... There are a few programs that can help, downloading in the background for you and possibly saving the result.

GET is the simplest possible client: it is invoked with one or more URLs on the command line and outputs download results to `stdout`. It is not of much use for batch download, but can be used to track down problems with webservers. See also **HEAD**

Wget (`http://www.wget.org/`) performs non-interactive download of files from the Web. It supports HTTP, HTTPS, and FTP protocols, as well as retrieval through HTTP proxies. It can parse downloaded files for links and recursively download all linked pages (be careful, since this can lead to unwanted massive download! see `man wget` for recursive downloading options)

cURL (`http://curl.haxx.se/`) is a client to get documents/files from or send documents to a server, using any of the supported protocols (HTTP, HTTPS, FTP, GOPHER, DICT, TELNET, LDAP or FILE). The command is designed to work without user interaction or any kind of interactivity. **curl** offers a busload of useful tricks like proxy support, user authentication, ftp upload, HTTP post, SSL (https:) connections, cookies, file transfer resume and more. There is a *large* help for curl in `man curl`

9.2 Setting up a web proxy

The use of a web proxy can improve both tranfer speed and response time when accessing many popular web sites, while reducing the total traffic on SISSA external link. This is achieved keeping a local copy of frequently visited pages. The local copy is validated before being served to the client, i.e. compared to a set of refresh patterns to ensure the information is still up-to-date.

You should configure your browser to use the proxy with the automatic configuration script provided at `http://proxy.sissa.it/cgi-bin/proxy.pac`. The script will configure your browser for using the proxy farm, with load balancing and high availability in case of failure of one of the proxy boxes.

Browser-specific information

- **Netscape**, **Mozilla**
  1. Open Edit → Preferences → Advanced → Proxy
  2. Select «Automatic proxy configuration URL»
  3. Fill in the URL provided above
- **Galeon**

1. Open Settings → Advanced → Network
2. Select «Automatic proxy configuration»
3. Fill in the URL provided above
4. Click on «Apply»

- **Internet Explorer**

  1. Open Tools → Internet Options
  2. Select the Connections tab
  3. Select the LAN Settings button
  4. Fill in the URL provided above

- **Konqueror**

  1. Open Settings → Configure Konqueror
  2. Select the Proxy icon
  3. Select Use Proxy
  4. Select Specified script file
  5. Fill in the URL provided above
  6. Click on «Apply»

- **Lynx**

  (a) if you are using bash or zsh: `export http_proxy="http://proxy.sissa.it:3128/"` (you can put this in your .bashrc)

  (b) if you are using csh or tcsh: `setenv http_proxy "http://proxy.sissa.it:3128/"` (you can put this in your .tcshrc)

  Since **lynx** does not support proxy autoconfiguration, this setup provides only fault tolerance, but no load balancing. This setup works with **GET**, **wget** and **cURL** too.

If you encounter any problem, please fill in the error reporting forms provided with proxy error messages with as much information as you can. Your reports will help us to track down existing problems and improve service quality.

Proxy Q&A

Q: I have 20Gb of free space on my workstation available for browser cache; I never use another browser, nor any other workstation. Why should I set up my browser to use a proxy?

A: There are at least three good reasons:

1. the proxy cache is shared among all users: this means that you can benefit from downloads made by others at SISSA, and they can benefit from yours
2. the proxy software is entirely devoted to caching and uses better algorithms than your browser to find out if a cached page is still fresh or should be regarded as stale, so you will get a higher hit rate with lower risk of being served stale contents
3. most browsers are tuned to work with small to mid-size caches (tipically of the order of 10Mb) and tend to perform badly with very large caches: the time needed to search the local cache can be larger than the one needed to download the page itsef; on older machines the browser can actually crash or even make the whole system unstable

Q: I'm going to browse a SSL-enabled site, and I do not want reserved information to be cached. Should I disable the proxy in my browser?

A: encrypted connections are proxied but not cached – in fact caching encrypted information is useless since nobody will be able to decrypt it but the intended recipient. However some sites serve only sensitive informations over SSL, while other page components (e.g. navigation bars) are served unencrypted and these can be usefully cached, so you can safely use the proxy.

9.3 File transfer

Web browsers can use the file transfer protocol for anonymous ftp, as the **wget** and **curl** described above. They can also perform non-anonymous ftp, but the password is sent as plain text over the network, and sometimes it is shown on screen too! You should better use **sftp** and **scp** that use encrypted connections for both authentication and data transfer. While **scp** is non-interactive and lets you tranfer files with a single command line, **sftp** is an interactive client with the same user interface of plain old **ftp** client.

scp and sftp quick reference

`scp path/file user@host:path/file`	*copy local file to remote host*
`scp user@host:path/file path1/file1`	*copy remote file to local filesystem*
`scp u1@h1:path1/file1 u2@h2:parh2/file2`	*copy between remote hosts*
`scp -r ...`	*copy recursively entire directories*
`scp -C ...`	*enable compression – this will speed up some tranfers on slow lines*
`sftp user@host`	*start the sftp client and login to specified host*
`sftp -C ...`	*start the sftp client enabling compression*

Within the **sftp** client all standard **ftp** commands are available, see `man sftp` or the online help typing `help` at the `sftp>` prompt. Remember that wildcards are expanded by the (local) shell, unless they are single quoted so `scp file*.tex user@host:/tmp` works as expected, but you need to write `scp user@host:'/home/user/file*.tex' /tmp` to perform the opposite: without the single quotes the shell would have locally expanded the `file*.tex` expression, probably resulting in some error or at least in something different from what you wanted.

10 Other software

10.1 Compilers and libraries

The standard compilers for Linux machines are **gcc** (C/C++) and **g77** (fortran77). Extensive help is available from `info gcc`, `info g77`.

Several spcialized compilers are available. Refer to `http://www.sissa.it/comp/online-manuals/index.html`.

10.2 TeX, LaTeX and related programs

The TeX/LaTeX suite is available on most machines, along with its (large) help system. The main index for LaTeX help is at `file:///usr/share/texmf/doc/newhelpindex.html`. Please remember that all printers at SISSA use *A4* paper, so documents should be formatted accordingly (and not using the default *letter* format).

10.3 CD burning software

The classical CD burning tools for Linux are **mkisofs** and **cdrecord**. Although they provide fine-grain control over the burning process, their use is not actually that simple. A better starting point for beginners is **xcdroast**, that provides a simpler (but somehow limited) graphical interface. Newer workstations feature the **k3b** CD burning software that provides both a simpler interface for beginners and a wider capability for expert users.

11 Help

11.1 Who, When, Where

For information on the use of SISSA research and computing facilities, please contact staff by e-mail, by phone or directly. The Computer Staff offices are located at the ground floor.

Accesses to Staff rooms as well as the telephone calls are limited **from 9:30 to 11:30** in the morning and **from 15:00 to 16:00** in the afternoon. Outside these periods you are kindly invited to use the e-mail.

Identification of the problem is a prerequisite for an effective solution. The following scheme will help you in the choice of the right assistant.

If your problem is...	*Assistant (Substitute)*
a Linux workstation	**M. Picek, P. Accerboni** (L. Urgias)
a Windows (NT/2000) computer	**A. Tomicich** (F. Lonzar)
Hardware failure	**F. Lonzar** (A. Tomicich)
Computer network & Internet	**M. Talpo** (D. Brunato)
E-mail	**D. Brunato** (L. Urgias, M. Talpo)
Calculus machines (OpenMosix, MPI)	**P. Calucci** (D. Brunato)
Programming & compilers	**L. Urgias, M. Talpo** (D. Brunato)
SISSA web pages	**M. Zanello** (L. Urgias, D. Brunato)

Please contact the substitute *only* in the case of absence of the assistant. If you are unable to identify the problem, send an e-mail with a description to `helpdesk@sissa.it`.

11.2 How

When asking for help please be prepared to answer the following questions:

1. what are the sympthoms of the problem?
   - bad answer: «I can't log in»
   - good answer: «when I try to log in via ssh, I get an error message about server key having been changed»
2. which machines are involved?
   - bad answer: «the main cluster»
   - good answer: «I was trying to log in from my home PC running BrandNewOS to `veryoldserver.sissa.it`»
3. what applications/commands/actions/files are involved?
   - bad answer: «my favourite editor»
   - good answer: «`nedit` 5.3-1 with Motif 2.1.30, running on RedHat 8.0»
4. is the problem reproducible? how?
5. what were you doing just before the problem arose?
6. have you recently changed your password/preferences/configuration files?
7. do you know if this applies to other users as well?
8. *(if you have solved or worked around it)* how did you do?

Personnel addresses and working hours

Name	E-mail	Phone	Office location	Working hours
Pietro Accerboni	`ace@sissa.it`	485	ground floor, room 29	8:30-14:30 (17:30 tue, thu)
Mirna Avezzù	`avezzu@sissa.it`	536	ground floor, room 30	8:30-14:30 (17:30 wed, fri)
Davide Brunato	`brunato@sissa.it`	538	ground floor, room 28	8:30-14:30 (17:30 wed, fri)
Piero Calucci	`calucci@sissa.it`	496	ground floor, room 27	8:30-14:30 (17:30 tue, fri)
Fabio Lonzar	`lonzar@sissa.it`	540	ground floor, room 26	8:30-14:30 (17:30 mon, tue)
Marina Picek	`marina@sissa.it`	415	ground floor, room 27	8:30-14:30 (17:30 mon, tue)
Marco Talpo	`talpo@sissa.it`	506	ground floor, room 30	8:30-14:30 (17:30 mon, wed)
Andrea Tomicich	`tomicich@sissa.it`	537	ground floor, room 26	8:30-14:30 (17:30 tue, thu)
Luisa Urgias	`urgias@sissa.it`	539	ground floor, room 29	8:30-13:30
Maurizio Zanello	`zanello@sissa.it`	542	ground floor, room 28	8:30-14:30 (17:30 tue, thu)