# Basic Usage (Command Line Interface, Shell Scripts)

## Important info:

Large parts of this script were taken from the documentation of the Hamburg HPC Competence Center (HHCC). Please visit their website for more details on the Use of the Command Line Interface or about Using Shell Scripts.

## Description:

- You will learn about the book “The Linux Command Line” by William Shotts which we recommend for learning the command line (basic level)
- You will learn to login remotely (basic level)
- You will learn to use text editors (first steps without the need to consult the book) (basic level)
- You will learn to handle scripting tasks that are often needed in batch scripts: manipulating filenames, temporary files, tracing command execution, error handling, trivial parallelization (basic level)

# Use of the Command Line Interface

Users interact with an HPC system through a command line interface (CLI), there are no graphical user interfaces. The command line is used for interactive work and for writing shell scripts that run as batch jobs.

Using a command line interface is a good way to interact very efficiently with a computer by just typing on a keyboard. Fortunately this is especially true for Unix-like operating systems like Linux, because for the typically Linux based cluster systems graphical user interfaces (GUIs) are rarely available. In the Unix or Linux world a program named shell is used as a command language interpreter. The shell executes commands it reads from the keyboard or – more strictly speaking – from the standard input device (e.g. mapped to a file).

The shell can also run programs named shell scripts, e.g. to automate the execution of several commands in a row.

The command line is much more widely used than in HPC. Accordingly, much more has been written about the command line than on HPC. Instead of trying to write yet another text on the command line we would like to refer the reader to the very nice book The Linux Command Line by

William Shotts. The book can be read online. A pdf version (here: 19.01) is also available.

As a motivation to start reading it, we quote a few lines from the Introduction of version 19.01 of the book:

- It's been said that "graphical user interfaces make easy tasks easy, while command line interfaces make difficult tasks possible" and this is still very true today.
- …, there is no shortcut to Linux enlightenment. Learning the command line is challenging and takes real effort. It's not that it's so hard, but rather it's so vast.
- And, unlike many other computer skills, knowledge of the command line is long lasting.
- Another goal is to acquaint you with the Unix way of thinking, which is different from the Windows way of thinking.

The book also introduces the secure shell (ssh). The secure shell is a prerequisite for using HPC systems, because it is needed to log in, and to copy files from and to the system. We also mention the very first steps with text editors.

# Remote login

## Login nodes

To get access to a cluster system, a terminal emulator program is used to remotely connect to a cluster node (possibly belonging to a group of such nodes) which authenticates the user and grants access to the actual cluster system. Such login nodes are also named gateway nodes or head nodes.

## SSH (Secure Shell) and Windows Equivalents

The ssh program provides a secured and encrypted communication for executing commands on a remote machine like building programs and submitting jobs on the cluster system. Windows users can use third-party software like putty or MobaXterm to establish ssh connections to a cluster system. Meanwhile a quite mature OpenSSH port (beta version), i.e. a collection of client/server ssh utilities, is also available for Windows.

Another very useful option is to run Linux (e.g. Ubuntu or openSUSE) on a Windows computer in a virtual machine (VM) using a hypervisor like VMware Workstation Player, VirtualBox, or Hyper-V.

## SFTP Clients with Graphical User Interfaces

To make it easier to view files on the server as well as move files between your computer and the server, it possible to use a SFTP Client. SSH File Transfer Protocol (also Secure File Transfer

Protocol, or SFTP) is a network protocol that provides file access, file transfer, and file management over any reliable data stream.

There are numerous user-friendly tools providing a GUI, such as WinSCP for Windows or FileZilla for Windows, Linux and Mac OS X (please make sure to decline any possible adware when using the installer).

Other options for more advanced Linux users are SSHFS, a tool that uses SSH to enable mounting of a remote filesystem on a local machine, or the use of the common Gnome Nautilus File Browser. Please visit the links for further information.

# Text editors

On an HPC-cluster one typically works in a terminal mode (or text mode in contrast to a graphical mode), i.e. one can use the keyboard but there is neither mouse support nor graphical output. Accordingly, text editors have to be used in text mode as well. For newcomers this is an obstacle. As a workaround, or for editing large portions, one can use the personal computer, where a graphical mode is available, and copy files to the cluster when editing is completed. However, copying files back and forth can be cumbersome if edit cycles are short: for example, if one is testing code on a cluster, where only small changes are necessary, using an editor directly on the cluster is very helpful.

## Text user interface

Classic Unix/Linux text editors are vi/vim and GNU Emacs. Both are very powerful but their handling is not intuitive. The least things to know are the key strokes to quit them:

- vi: `<esc>:q!` (quit without saving)
- vi: `<esc>ZZ` (save and quit)
- emacs: `<cntl-x><cntl-c>`

GNU nano is a small text editor that is more intuitive, in particular, because the main control keys are displayed with explanations in two bars at the bottom of the screen. The exit key is:

- nano: `<cntl-x>`.

## Graphical user interface

In addition to text user interfaces vim and GNU Emacs have graphical interfaces. On an HPC-cluster it is possible to use graphical interfaces if X11 forwarding is enabled (see -X option of the ssh command). Two other well know text editors, that you might find on your cluster, are gedit and kate.

However, X11 forwarding can be annoyingly slow if the internet connection is not good enough.

A trick for advanced users is of course again to mount file systems from a cluster with SSHFS. Then one can transparently use one's favourite text editor on the local computer for editing files on the remote cluster.

# Using shell scripts

Shell scripts are used to store complicated commands or to automate tasks. A simple script consists of one or a few commands that appear exactly like on the interactive command line. Since the shell is also a programming language, scripts can execute more complicated processes. On HPC systems scripting is needed for creating batch jobs. Here, some scripting tasks are explained that are useful for writing batch scripts.

## Manipulating filenames

Handling filenames translates to character string processing. The following table shows some typical examples:

| action | command | result |
|---|---|---|
| initialization | a=foo | a=foo |
| | b=bar | b=bar |
| concatenation | c=$a/$b.c | c=foo/bar.c |
| | d=${a}_$b.c | d=foo_bar.c |
| get directory | dir=$(dirname $c) | dir=foo |
| get filename | file=$(basename $c) | file=bar.c |
| remove suffix | name=$(basename $c .c) | name=bar |
| | name=${file%.c} | name=bar |
| remove prefix | ext=${file##*.} | ext=c |

*Recommendation: Never use white space in filenames!* This is error prone, because quoting becomes necessary, like in: `dir=$(dirname „$c“)`.

## Temporary files

There are three issues with temporary files: choice of the directory to write them, unique names and automatic deletion.

Assume that a batch job shall work in a temporary directory. Possibly, the computing center provides such a directory for every batch job and deletes that directory when the jobs ends. Then one can just use that directory. If this is not the case one can proceed like this.

- Choose a file system (or top directory) to work in. The classic directory for this purpose is `/tmp`. However, this is not a good choice on (almost all) HPC clusters, because `/tmp` is probably too small on diskless nodes. You should find out for your system, which file system is well suited. There can be local file systems (on nodes that are equipped with local disks) or global file systems. Let us call that filesystem `/scratch` and set:

```
    top_tmpdir=/scratch
```

- A sub-directory with a unique name can be generated with the `mktemp` command. `mktemp` generates a unique name from a template by replacing a sequence of `X`s by a unique value. It prints the unique name such that it can be stored in a variable. For easy identification of your temporary directories you can use your username (which is contained in the variable `$USER`) in the template, and set:

```
    my_tmpdir=$(mktemp -d "$top_tmpdir/$USER.XXXXXXXXXXX")
```

- The next line in our example handles automatic deletion. Wherever the script exits our temporary directory will be deleted:

```
    trap "rm -rf $my_tmpdir" EXIT
```

- Now we can work in our temporary directory:

```
    cd $my_tmpdir
    ...
```

## Tracing command execution

There are two shell settings for tracing command execution. After `set -v` all commands are printed as they appear literally in the script. After `set -x` commands are printed as they are being executed (i.e. with variables expanded). Both settings are also useful for debugging.

## Error handling

There are two shell settings that can help to handle errors.

The first setting is `set -e` which makes the script exit immediately if a command exits with an error (non-zero) status. The second setting is `set -u` which makes the script exit if an undefined variable is accessed (this is also useful for debugging).

Exceptions can be handled in the following ways. If `-e` is set an error status can be ignored by using the *or* operator `||` and calling the `true`command which is a no-op command that always succeeds:

```
  command_that_could_go_wrong || true
```

If `-u` is set a null value can be used if variable that is unset is accessed (the braces and the dash after `_set` do the job):

```
if [[ ${variable_that_might_not_be_set-} = test_value ]]
then
    ...
fi
```

# Information about the filesystems

Phoenix has two different filesystems: /home and /work.

When you connect to Phoenix, the current dictionary is /home/username.
The /home-folder is a Network File System, a backup is performed every day.
Each user has a quota of 100 GB, to check how much of that is occupied you can use the command:

```
quota -vs
```

Each user also has another folder: /work/username.
The /work-folder is a BeeGFS, a filesystem developed and optimized for high-performance computing.
For all your calculations you should always read from and write to your /work-folder, because in comparison to the /home-folder it doesnot have problems with multiple nodes writing to the filesystem. There are no backups for the /work-folder, so you should copy important information after the calculations to the /home-folder.
The quota of the /work-folder is 5 TB, to check how much is occupied you can use the command:

```
beegfs-ctl --getquota --uid $USER
```

# File transfer from your local system to the Phoenix cluster

Files can be transferred from your local system to the Phoenix remote system in a number of ways. There are two common options: `scp` and `rsync`. In general `rsync` is the better option.

One option is to use the `Secure Copy Protocoll (SCP)`. To use this, you have to open a terminal on your local system. The following command copies the file myfile from your local system to a remote directory on the Phoenix:

```
scp path/to/myfile username@phoenix.hlr.rz.tu-bs.de:/path/to/destinationfolder
```

For copying a folder, `-r` has to be included:

```
scp path/to/myfolder -r username@phoenix.hlr.rz.tu-bs.de:/path/to/destinationfolder
```

If the target file does not yet exist, an empty file with the target file name is created and then filled with the source file contents. If copying a source file to a target file which already exists, scp will replace the whole contents of the target file; this will take longer then needed and in this case we recommend the usage of rsync.

`Rsync` is another option. Rsync can synchronize files and directories, it only transfers the differences between the source and the destination. Therefore it is recommended to use rsync for files and folders that already exist on the Phoenix but changed on your local system. It is also recommended when you tried to copy something to the Phoenix but due to a loss of internet connection, it did not finish. In this case only a part of the files are transferred, to copy only the missing files, using `rsync` is the best choice.

```
rsync -avz path/to/file username@phoenix.hlr.rz.tu-bs.de:/path/to/destinationfolder
```

For both ways there is a variety of flags that can be looked up in the corresponding man pages.

# File transfer from remote system to the Phoenix cluster

To transfer files from a remote system, it is necessary that you login via ssh on the remote server where the files are located (e.g. a server of the institute). Then start a `screen session` with the command:

```
screen
```

Then it is possible to just use the commands described above. The advantage of launching the copying-process in a screen session is, that you can disconnect from the remote server and the process will still be running. If you don't use a screen session and the SSH session to the remote server terminates (e.g. you disconnect yourself or the internet connection drops), then the copying-process will be terminated as well.

---

Revision #5
Created 2024-03-12 13:26:11 UTC by Michael Giemsa
Updated 2024-10-30 13:27:46 UTC by Matthias Franz