Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
Consiglio Nazionale delle Ricerche

# Linux Operation System as a base for Spatial Ecology Computing

Linux is a generic term refering to Unix-like computer operating systems based on the Linux kernel. Their development is one of the most prominent examples of free and open source software collaboration; typically all the underlying source code can be used, freely modified, and redistributed, both commercially and non-commercially, by anyone under licenses such as the GNU.

In this site an introduction will be given to the Unix/Linux Shell using Bash language to manipulate data rather than interacting with/setting the operation system. The final aim is to build a stand-alone implementation / processes that include a combination of bash/R/AWK/gnuplot commands that can be run several times using the features of each software. In this part of the training site we provide various examples of bash commands reported in this Unix/Linux Command Reference.

In the jupyter-notebook you can call/use bash language by using two symbols:

```
%%bash
bash-command
```

before the bash commands, or

```
! bash-command
```

followed by the bash commands

## Bash language syntax

The object of this document is the use of Bash language to explore and manipulate files rather than to set/interact with the operation system. You can read and follow jupter-notebook or you can copy the commands included in the frames part of this document and paste them into an interactive Bash shell. Once you have familiarity with the general commands of Bash you can

further advance in learning bash with online manuals and guides. There is a large variety of documentation available at: http://www.linux.org/lessons/advanced/x1110.html http://tldp.org/LDP/abs/html/

The best way is just to try each command using a file, and/or search on the Internet for more examples and deeper explanations.

**Searching for a command, getting help**

In a shell window (the terminal) the following prompt is written:

```
user@pc_name:directrory$
```

after the $ you are able to insert the command Command syntax:

```
command [option] [file]
```

The square bracts "[ ]" identify an optional feature of the command. It can be inserted to retrieve more information or different setting of a command. To get a command for a specific action type "man -k thewordthatyouneed"

e.g. I want to search for a command able to count the line in a file

[1]:
```
! man -k count
```

```
acct (2)             - switch process accounting on or off
acct (5)             - process accounting file
argz_count (3)       - functions to handle an argz list
cksum (1)            - checksum and count the bytes in a file
CPU_COUNT (3)        - macros for manipulating CPU sets
CPU_COUNT_S (3)      - macros for manipulating CPU sets
error_message_count (3) - glibc error reporting functions
ibv_attach_counters_point_flow (3) - attach individual counter definition to ...
ibv_destroy_counters (3) - Create or destroy a counters handle
ibv_read_counters (3) - Read counter values
fincore (1)          - count pages of file contents in core
get_avphys_pages (3) - get total and available physical page counts
get_phys_pages (3)   - get total and available physical page counts
git-count-objects (1) - Count unpacked number of objects and their disk consu...
goa-daemon (8)       - GNOME Online Accounts Daemon
ibv_create_counters (3) - Create or destroy a counters handle
mlx5dv_dr_action_create_flow_counter (3) - Create devx flow counter actions
mlx5dv_ts_to_ns (3)  - Convert device timestamp from HCA core clock units to ...
pam_lastlog (8)      - PAM module to display date of last login and perform i...
pam_succeed_if (8)   - test account characteristics
pam_tally (8)        - The login counter (tallying) module
pam_tally2 (8)       - The login counter (tallying) module
pcre16_refcount (3)  - Perl-compatible regular expressions
pcre2_get_ovector_count (3) - Perl-compatible regular expressions (revised API)
pcre32_refcount (3)  - Perl-compatible regular expressions
pcre_refcount (3)    - Perl-compatible regular expressions
rdma-statistic (8)   - RDMA statistic counter configuration
sum (1)              - checksum and count the blocks in a file
systemd-bless-boot-generator (8) - Pull systemd-bless-boot.service into the i...
timer_getoverrun (2) - get overrun count for a POSIX per-process timer
userdel (8)          - delete a user account and related files
usermod (8)          - modify a user account
v.in.geonames (1grass) - Imports geonames.org country files into a vector poi...
v.qcount (1grass)    - Indices for quadrat counts of vector point lists.
v.vect.stats (1grass) - Count points in areas, calculate statistics from poin...
wc (1)               - print newline, word, and byte counts for each file
```

in the last lines you get:

“wc (1) - print newline, word, and byte counts for each file”

so the command “wc” is your command. To get information about a command type “man command” or info “command” e.g.

[2]:
```
! man wc
```

```
WC(1)                          User Commands                          WC(1)

N✓NA✓AM✓ME✓E
       wc - print newline, word, and byte counts for each file

S✓SY✓YN✓NO✓OP✓PS✓SI✓IS✓S
       w✓wc✓c [_✓O_✓P_✓T_✓I_✓O_✓N]... [_✓F_✓I_✓L_✓E]...
       w✓wc✓c [_✓O_✓P_✓T_✓I_✓O_✓N]... _✓-_✓-_✓f_✓i_✓l_✓e_✓s_✓0_✓-
_✓f_✓r_✓o_✓m_✓=_✓F

D✓DE✓ES✓SC✓CR✓RI✓IP✓PT✓TI✓IO✓ON✓N
       Print newline, word, and byte counts for each FILE, and a total line if
       more than one FILE is specified.  A word is a non-zero-length  sequence
       of characters delimited by white space.

       With no FILE, or when FILE is -, read standard input.

       The  options  below may be used to select which counts are printed, al-
       ways in the following order: newline, word,  character,  byte,  maximum
       line length.

       - -c✓c, -✓--✓-b✓by✓yt✓te✓es✓s
              print the byte counts

       - -m✓m, -✓--✓-c✓ch✓ha✓ar✓rs✓s
              print the character counts

       - -l✓l, -✓--✓-l✓li✓in✓ne✓es✓s
              print the newline counts

       - --✓-f✓fi✓il✓le✓es✓s0✓0-✓-f✓fr✓ro✓om✓m=_✓F
              read  input  from the files specified by NUL-terminated names in
              file F; If F is - then read names from standard input

       - -L✓L, -✓--✓-m✓ma✓ax✓x-✓-l✓li✓in✓ne✓e-✓-l✓le✓en✓ng✓gt✓th✓h
              print the maximum display width

       - -w✓w, -✓--✓-w✓wo✓or✓rd✓ds✓s
              print the word counts

       - --✓-h✓he✓el✓lp✓p display this help and exit

       - --✓-v✓ve✓er✓rs✓si✓io✓on✓n
              output version information and exit

A✓AU✓UT✓TH✓HO✓OR✓R
       Written by Paul Rubin and David MacKenzie.

R✓RE✓EP✓PO✓OR✓RT✓TI✓IN✓NG✓G B✓BU✓UG✓GS✓S
       GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
       Report wc translation bugs to <https://translationproject.org/team/>

C✓CO✓OP✓PY✓YR✓RI✓IG✓GH✓HT✓T
       Copyright © 2018 Free Software Foundation, Inc.   License  GPLv3+:  GNU
       GPL version 3 or later <https://gnu.org/licenses/gpl.html>.
       This  is  free  software:  you  are free to change and redistribute it.
       There is NO WARRANTY, to the extent permitted by law.

S✓SE✓EE✓E A✓AL✓LS✓SO✓O
       Full documentation at: <https://www.gnu.org/software/coreutils/wc>
       or available locally via: info '(coreutils) wc invocation'

GNU coreutils 8.30             September 2019                         WC(1)
```

# Input/Output redirect

## Running a command, saving a result

The symbols “>” are used to save the result of a command in a file. Instead “<” is used to retrieve information from a file. In these cases, using the informatics terminology we can use the expression ’standard input redirection” or and “standard output redirection”.

This page summarize the Standard Input and Output Redirection commonly used.

In this course we will mainly use the symbol “>”, “>>”, “<”. e.g.

[4]:
```
!ls
```

```
00_Setting_Colab_for_for_Spatial_Ecology_course.ipynb  02_pktools_osgeo.ipynb
01_gdal.ipynb                                          03_bash_osgeo.ipynb
02_pktools_colab.ipynb                                 geodata
```

[5]:
```
! ls > mylist.txt
```

[6]:
```
! more mylist.txt
```

```
00_Setting_Colab_for_for_Spatial_Ecology_course.ipynb
01_gdal.ipynb
02_pktools_colab.ipynb
02_pktools_osgeo.ipynb
03_bash_osgeo.ipynb
geodata
mylist.txt
```

[7]:
```
! ls >> mylist.txt
```

[8]:
```
! more mylist.txt
```

```
00_Setting_Colab_for_for_Spatial_Ecology_course.ipynb
01_gdal.ipynb
02_pktools_colab.ipynb
02_pktools_osgeo.ipynb
03_bash_osgeo.ipynb
geodata
mylist.txt
00_Setting_Colab_for_for_Spatial_Ecology_course.ipynb
01_gdal.ipynb
02_pktools_colab.ipynb
02_pktools_osgeo.ipynb
03_bash_osgeo.ipynb
geodata
mylist.txt
```

# Special Characters

Special characters, also called metacharacters, are a group of characters that have particular meanings in the bash language. Listed here are those used in the following scripts. Type the examples and try to get the meaning.

The asterisk “*” symbol identifies a string with one or more character

[9]:
```
! ls /dev/tty*
```

```
/dev/tty    /dev/tty23  /dev/tty39  /dev/tty54      /dev/ttyS10  /dev/ttyS26
/dev/tty0   /dev/tty24  /dev/tty4   /dev/tty55      /dev/ttyS11  /dev/ttyS27
/dev/tty1   /dev/tty25  /dev/tty40  /dev/tty56      /dev/ttyS12  /dev/ttyS28
/dev/tty10  /dev/tty26  /dev/tty41  /dev/tty57      /dev/ttyS13  /dev/ttyS29
/dev/tty11  /dev/tty27  /dev/tty42  /dev/tty58      /dev/ttyS14  /dev/ttyS3
/dev/tty12  /dev/tty28  /dev/tty43  /dev/tty59      /dev/ttyS15  /dev/ttyS30
/dev/tty13  /dev/tty29  /dev/tty44  /dev/tty6       /dev/ttyS16  /dev/ttyS31
/dev/tty14  /dev/tty3   /dev/tty45  /dev/tty60      /dev/ttyS17  /dev/ttyS4
/dev/tty15  /dev/tty30  /dev/tty46  /dev/tty61      /dev/ttyS18  /dev/ttyS5
/dev/tty16  /dev/tty31  /dev/tty47  /dev/tty62      /dev/ttyS19  /dev/ttyS6
/dev/tty17  /dev/tty32  /dev/tty48  /dev/tty63      /dev/ttyS2   /dev/ttyS7
/dev/tty18  /dev/tty33  /dev/tty49  /dev/tty7       /dev/ttyS20  /dev/ttyS8
/dev/tty19  /dev/tty34  /dev/tty5   /dev/tty8       /dev/ttyS21  /dev/ttyS9
/dev/tty2   /dev/tty35  /dev/tty50  /dev/tty9       /dev/ttyS22
/dev/tty20  /dev/tty36  /dev/tty51  /dev/ttyprintk  /dev/ttyS23
/dev/tty21  /dev/tty37  /dev/tty52  /dev/ttyS0      /dev/ttyS24
/dev/tty22  /dev/tty38  /dev/tty53  /dev/ttyS1      /dev/ttyS25
```

The questionmark “?” symbol identifies a a single character

[14]:
```
%%bash
ls /dev/tty?
```

```
/dev/tty0
/dev/tty1
/dev/tty2
/dev/tty3
/dev/tty4
/dev/tty5
/dev/tty6
/dev/tty7
/dev/tty8
/dev/tty9
```

The square brackets “[ ]” identify one of a single character listed

[15]:
```
! ls /dev/tty[2-4]
```

```
/dev/tty2  /dev/tty3  /dev/tty4
```

Curly brackets “{}” symbol identify one of a single string listed

[24]:
```
%%bash
ls /dev/{tty,loop}*
```

```
/dev/loop0
/dev/loop1
/dev/loop10
/dev/loop11
/dev/loop12
/dev/loop13
/dev/loop14
/dev/loop15
/dev/loop16
/dev/loop2
/dev/loop3
/dev/loop4
/dev/loop5
/dev/loop6
/dev/loop7
/dev/loop8
/dev/loop9
/dev/loop-control
/dev/tty
/dev/tty0
/dev/tty1
/dev/tty10
/dev/tty11
/dev/tty12
/dev/tty13
/dev/tty14
/dev/tty15
/dev/tty16
/dev/tty17
/dev/tty18
/dev/tty19
/dev/tty2
/dev/tty20
/dev/tty21
/dev/tty22
/dev/tty23
/dev/tty24
/dev/tty25
/dev/tty26
/dev/tty27
/dev/tty28
/dev/tty29
/dev/tty3
/dev/tty30
/dev/tty31
/dev/tty32
/dev/tty33
/dev/tty34
/dev/tty35
/dev/tty36
/dev/tty37
/dev/tty38
/dev/tty39
/dev/tty4
/dev/tty40
/dev/tty41
/dev/tty42
/dev/tty43
/dev/tty44
/dev/tty45
/dev/tty46
/dev/tty47
/dev/tty48
/dev/tty49
/dev/tty5
/dev/tty50
/dev/tty51
/dev/tty52
/dev/tty53
/dev/tty54
/dev/tty55
/dev/tty56
```

```
/dev/tty57
/dev/tty58
/dev/tty59
/dev/tty6
/dev/tty60
/dev/tty61
/dev/tty62
/dev/tty63
/dev/tty7
/dev/tty8
/dev/tty9
/dev/ttyprintk
/dev/ttyS0
/dev/ttyS1
/dev/ttyS10
/dev/ttyS11
/dev/ttyS12
/dev/ttyS13
/dev/ttyS14
/dev/ttyS15
/dev/ttyS16
/dev/ttyS17
/dev/ttyS18
/dev/ttyS19
/dev/ttyS2
/dev/ttyS20
/dev/ttyS21
/dev/ttyS22
/dev/ttyS23
/dev/ttyS24
/dev/ttyS25
/dev/ttyS26
/dev/ttyS27
/dev/ttyS28
/dev/ttyS29
/dev/ttyS3
/dev/ttyS30
/dev/ttyS31
/dev/ttyS4
/dev/ttyS5
/dev/ttyS6
/dev/ttyS7
/dev/ttyS8
/dev/ttyS9
```

# Quoting

You can prevent the shell from interpreting a metacharacter by placing a backslash “”. In this way the metacharacter become a normal character.

file1 will be copied to file?

[26]:
```
! cp mylist.txt mylist\?.txt
! ls
```

You can also insert the metacharacter between quotation marks.

[27]:
```
! ls /dev/"tt*"
```

```
ls: cannot access '/dev/tt*': No such file or directory
```

# Pipe

The pipe "|" metacharacter enables you to run a set of chained processes. To understand lets do an example creating a temporal file called tmp.txt and counting how many lines there are in the file.

[30]:
```
%%bash
ls /usr/bin > tmp.txt
wc -l tmp.txt
```

```
2227 tmp.txt
```

The same can be written

[31]:
```
! ls /usr/bin | wc -l
```

```
2227
```

without creating an intermediate file.