Unix Basics

Accessing a UNIX machine

Mac or LINUX

To log-in into the remote Linux shell, open terminal and type:

```
ssh -X <your_username>@<host_name>
```

host name is the remote server's domain name (e.g. srvUbuntu or sop.ase.ro)
You will be asked to enter the password, simply type it and press enter.

To copy files **To** the server run the following on your workstation or laptop:

```
scp -r <path_to_directory> <your_username>@<host_name>:
```

To copy files **From** the server run the following on your workstation or laptop:

```
scp -r <your_username>@<host_name>:<path_to_directory> .
```

Windows

1. Open Putty and select ssh. *Download PuTTY if you do not have it.*
2. Provide the host name (the remote server's domain name) and session name
3. Enter your identity information
4.

username:	*your username*
password:	*your password* **Nothing will show-up, simply type the password and press enter.**

5. ~~Setup for graphics emulation. *Download and install Xming if you do not have it.*~~
6. Use WinSCP or FileZilla for file exchange. Download and install WinSCP or FileZilla if you do not have it.

Basics

Command-Line Syntax for this Manual

Remember the UNIX/LINUX command line is case sensitive!
The hash (pound) sign "**#**" indicates end of a command and the start of a comment.
The notation < . . . > refers to variables and file names that need to be specified by the user. The symbols < and > need to be excluded.

Orientation

Viewing and changing the present working directory:

```
pwd                # Get full path of the present working directory (same as "echo $HOME")
 ls                # Content of pwd
 ls -l             # Similar as ls, but provides additional info on files and directories
```

```
ls -a             # Includes hidden files (.name) as well
ls -R             # Lists subdirectories recursively
ls -t             # Lists files in chronological ordercd <dir_name>
cd                # Brings you to the highest level of your home directory.
cd ..             # Moves one directory up
cd ../../         # Moves two directories up (and so on)
cd -               # Go back to you were previously (before the last directory change)
```

The tilde symbol (~) gets interpreted as the path to your home directory. This will happen anywhere on the command line:

```
echo ~            # View the full (complete) path of your home
find ~            # List all your files (including everything in sub-directories)
ls ~             # List the top level files of your home directory
du -sch ~/*       # Calculate the file sizes in your home
```

Viewing file info, user, and host:

```
stat <file-name>  # Last modification time stamps, permissions, and size of a filewhoami
hostname          # Shows on which machine you are (same as "echo $HOSTNAME")
```

Files and directories

```
mkdir <dir_name>   # Creates specified directory
rmdir <dir_name>   # Removes empty directory
rm <file_name>     # Removes file name
rm -r <dir_name>   # Removes directory including its content, but asks for confirmation,'f' arg
ument turns confirmation off
cp <name> <path>   # Copy file/directory as specified in path (-r to include content in directo
ries)
mv <name1> <name2> # Renames directories or files
mv <name> <path>   # Moves file/directory as specified in path
```

Copy and paste

The methods differ depending where you are.

- In a **command line** environment:

Cut last word with keyboard only

- `Ctrl+w #Press multiple times to cut more than one word`

Paste with keyboard only

```
Ctrl+y
```

In a non-command line **desktop** environment:

Copy

```
Ctrl+c
```

Paste

```
Ctrl+v
```

Command line <-> **desktop** exchange:
Copy text out of the command line and into the **desktop:**

```
Shift+Ctrl+c        or        Apple+c
```

Paste text from the **desktop** into the command line**:**

```
Shift+Ctrl+v        or        Apple+v
```

Handy shortcuts

Anywhere in Command Line:

up(down)_key – scrolls through command history

```
history   # shows all commands you have used recently
```

Auto Completion:

<something-incomplete> TAB – completes program_path/file_name

Taking control over the cursor (the pointer on the command line):

```
Ctrl+a    # cursor to beginning of command line
Ctrl+e    # cursor to end of command line
Ctrl-w    # Cut last word
Ctrl+k    # cut to the end of the line
Ctrl+y    # paste content that was cut earlier (by Ctrl-w or Ctrl-k)
```

When specifying file names:

“.” (dot) – refers to the present working directory
“~” (Tilda) or “~/” – refers to user’s home directory

Unix Help

```
man <something> # general help (press the 'q' key to exit)
man wc          # manual on program 'word count' wc
wc --help       # short help on wc
soap -h         # for less standard programs
```

Universally available Linux commands, with detailed examples and explanations: http://www.linuxconfig.org/linux-commands

Finding Things

Finding files, directories and applications

```
find -name "*pattern*"            # searches for *pattern* in and below current directory
find /usr/local -name "*blast*"   # finds file names *blast* in specfied directory
find /usr/local -iname "*blast*"  # same as above, but case insensitive
```

Additional useful arguments: -user <user name>, -group <group name>, -ctime <number of days ago changed>

```
find ~ -type f -mtime -2   # finds all files you have modified in the last two days
locate <pattern>           # finds files and dirs that are written into update file
which <application_name>   # location of application
whereis <application_name> # searches for executeables in set of directories
dpkg -l | grep mypattern   # find Debian packages and refine search with grep pattern
```

Finding things in files

```
grep pattern file           # provides lines in 'file' where pattern 'appears',
# if pattern is shell function use single-quotes:
'>'grep -H pattern          # -H prints out file name in front of pattern
grep 'pattern' file | wc    # pipes lines with pattern into word count wc
 # wc arguments: -c: show only bytes, -w: show only words,
 # -l: show only lines; help on regular expressions:
man 7 regex or man perlrefind /home/my_dir -name '*.txt' | xargs grep -c ^.*  # counts line num
bers on many
 # files and records each count along with individual file
 # name; find and xargs are used to circumvent the Linux
 # wildcard limit to apply this function on thousands of files.
```

Permissions and Ownership

List directories and files

```
ls -al   # shows something like this for each file/dir: drwxrwxrwx
          # d: directory
          # rwx: read write execute
          # first triplet: user permissions (u)
          # second triplet: group permissions (g)
          # third triplet: world permissions (o)
```

Assign write and execute permissions to user and group

```
chmod ug+rx my_file
```

To remove all permissions from all three user groups

```
chmod ugo-rwx my_file
# '+' causes the permissions selected to be added
# '-' causes them to be removed
# '=' causes them to be the only permissions that the file has.
 chmod +rx public_html/ or $ chmod 755 public_html/ # Example for number system:
```

Change ownership

```
chown <user> <file or dir>          # changes user ownership
chgrp <group> <file or dir>         # changes group ownership
chown <user>:<group> <file or dir> # changes user & group ownership
```

Useful Unix Commands

```
df              # disk space
free -g         # memory info in Megabytes
uname -a        # shows tech info about machine
bc              # command-line calculator (to exit type 'quit')
wget <ftp>      # file download from web
/sbin/ifconfig # give IP and other network info
ln -s original_file new_file # creates symbolic link to file or directory
du -sh          # displays disk space usage of current directory
du -sh *        # displays disk space usage of individual files/directories
du -s * | sort -nr # shows disk space used by different directories/files sorted by size
```

Process Management

General

```
top               # view top consumers of memory and CPU (press 1 to see per-CPU statistics)
who               # Shows who is logged into system
w                 # Shows which users are logged into system and what they are doing
ps                # Shows processes running by user
ps -e             # Shows all processes on system; try also '-a' and '-x' arguments
ps aux | grep <user_name> # Shows all processes of one user
ps ax --tree      # Shows the child-parent hierarchy of all processes
ps -o %t -p <pid> # Shows how long a particular process was running. # (E.g. 6-04:30:50 means 6
days 4 hours ...)
Ctrl z <enter>    # Suspend (put to sleep) a process
```

```
fg                # Resume (wake up) a suspended process and brings it into foreground

bg                # Resume (wake up) a suspended process but keeps it running in the background
.

Ctrl c            # Kills the process that is currently running in the foreground

kill <process-ID> # Kills a specific process

kill -9 <process-ID> # NOTICE: "kill -9" is a very violent approach. It does not give the proce
ss any time to perform cleanup procedures.

kill -l           # List all of the signals that can be sent to a proccess

kill -s SIGSTOP <process-ID> # Suspend (put to sleep) a specific process

kill -s SIGCONT <process-ID> # Resume (wake up) a specific process

renice -n <priority_value> # Changes the priority value, which range from 1-19,the higher the v
alue the lower the priority, default is 10.
```

Text Viewing

```
more <my_file>  # views text, use space bar to browse, hit 'q' to exit

less <my_file>  # a more versatile text viewer than 'more', 'q' exits, 'G' end of text, 'g' beg
inning, '/' find forward, '?' find backwards

cat  <my_file>  # concatenates files and prints content to standard output
```

Text Editors

Vi and Vim

Non-graphical (terminal-based) editor. Vi is guaranteed to be available on any system. Vim is the improved version of vi.

Emacs

Non-graphical or window-based editor. You still need to know keystroke commands to use it. Installed on all Linux distributions and on most other Unix systems.

Nano

A simple terminal-based editor which is default on modern Debian systems.

The Unix Shell

When you log into UNIX/LINUX system, then is starts a program called the Shell. It provides you with a working environment and interface to the operating system. Usually there are several different shell programs installed. The shell program `bash` is one of the most common ones.

```
finger <user_name> # shows which shell you are using

chsh -l # gives list of shell programs available on your system (does not work on all UNIX vari
ants)

<shell_name> # switches to different shell
```

STDIN, STDOUT, STDERR, Redirections, and Wildcards

By default, UNIX commands read from standard input (STDIN) and send their output to standard out (STDOUT). You can redirect them by using the following commands:

```
<beginning-of-filename>*         # * is wildcard to specify many files
```

```
ls > file                          # prints ls output into specified file
command < my_file                  # uses file after '<' as STDIN
command >> my_file                 # appends output of one command to file
command | tee my_file              # writes STDOUT to file and prints it to screen
command > my_file; cat my_file     # writes STDOUT to file and prints it to screen
command > /dev/null                # turns off progress info of applications by redirecting their output to /dev/null
grep my_pattern my_file | wc       # Pipes (|) output of 'grep' into 'wc'
grep my_pattern my_non_existing_file 2 > my_stderr # prints STDERR to file
```

Useful shell commands

```
cat <file1> <file2> > <cat.out>      # concatenate files in output file 'cat.out'
paste <file1> <file2> > <paste.out>  # merges lines of files and separates them by tabs (useful for tables)
cmp <file1> <file2>                  # tells you whether two files are identical
diff <fileA> <fileB>                 # finds differences between two files
head -<number> <file>                # prints first lines of a file
tail -<number> <file>                # prints last lines of a file
split -l <number> <file>             # splits lines of file into many smaller ones
csplit -f out fasta_batch "%^>%" "/^>/" "{*}" # splits fasta batch file into many files at '>'
sort <file>                          # sorts single file, many files and can merge (-m) them, -b ignores leading white space
sort -k 2,2 -k 3,3n input_file > output_file # sorts in table col 2 alphabetically and col 3 numerically, '-k' for column, '-n' for numeric
sort input_file | uniq > output_file # uniq command removes duplicates and creates file/table with unique lines/fields
join -1 1 -2 1 <table1> <table2>     # joins two tables based on specified column numbers
 # (-1 file1, 1: col1; -2: file2, col2). It assumes that join fields are sorted. If that is not the case, use the next command:
sort table1 > table1a; sort table2 > table2a; join -a 1 -t "`echo -e '\t'`" table1a table2a > table3  # '-a <table>' prints all lines of specified table!
 # Default prints only all lines the two tables have in
 # common. '-t "`echo -e '\t'`" ->' forces join to
 # use tabs as field separator in its output. Default is
 # space(s)!!!
cat my_table | cut -d , -f1-3        # cut command prints only specified sections of a table,
 # -d specifies here comma as column separator (tab is
 # default), -f specifies column numbers.
```

Screen

- A Visual Introduction to Screen

- http://blogamundo.net/code/screen/
- http://fosswire.com/post/2008/08/video-tutorial-getting-started-with-gnu-screen/

Starting a New Screen Session

```
screen                 # Start a new session

screen -S <some-name>  # Start a new session and gives it a name
```

Commands to Control Screen

```
Ctrl-a d #  Detach from the screen session

Ctrl-a c # Create a new window inside the screen session

Ctrl-a Space # Switch to the next window

Ctrl-a a # Switch to the window that you were previously on

Ctrl-a " # List all open windows. Double-quotes " are typed with the Shift key

Ctrl-d or type exit # Exit out of the current window. Exiting form the last window will end the screen session

Ctrl-a [ # Enters the scrolling mode. Use Page Up and Page Down keys to scroll through the window. Hit the Enter key twice to return to normal mode.
```

Attaching to Screen Sessions

From any computer, you can attach to a screen session after SSH-ing into a server.

```
screen -r              # Attaches to an existing session, if there is only one

screen -r              # Lists available sessions and their names, if there are more then one session running

screen -r <some-name>  # Attaches to a specific session

screen -r <first-few-letters-of-name> # Type just the first few letters of the name & you will attach to the session you need
```

Destroying Screen Sessions

1. Terminate all programs that are running in the screen session. The standard way to do that is:

```
Ctrl-c
```

2. Exit out of your shell.

```
exit
```

3. Repeat steps 1 and 2 until you see the message:
`[screen is terminating]`
There may be programs running in different windows of the same screen session. That's why you may need to terminate programs and `exit` shells multiple time.

Simple One-Liner Shell Scripts

- Web page for ~~script download~~.

Renames many files *.old to *.new. To test things first, replace 'do mv' with 'do echo mv'.

```
for i in *.input; do mv $i ${i/\.old/\.new}; done
for i in *\ *; do mv "$i" "${i// /_}"; done # Replaces spaces in files by underscores
```

Run an application in loops on many input files.

```
for i in *.input; do ./application $i; done
```

Run `fastacmd` from BLAST program in loops on many *.input files and create corresponding *.out files.

```
for i in *.input; do fastacmd -d /data/../database_name -i $i > $i.out; done
```

Run SAM's target99 on many input files.

```
for i in *.pep; do target99 -db /usr/../database_name -seed $i -out $i; done
```

Search in many files for a pattern and print occurrences together with file names.

```
for j in 0 1 2 3 4 5 6 7 8 9; do grep -iH <my_pattern> *$j.seq; done
```

Example of how to run an interactive application (tmpred) that asks for file name input/output.

```
for i in *.pep; do echo -e "$i\n\n17\n33\n\n\n" | ./tmpred $i > $i.out; done
```

Run BLAST2 for all *.fasa1/*.fasta2 file pairs in the order specified by file names and write results into one file.

```
for i in *.fasta1; do blast2 -p blastp -i $i -j ${i/_*fasta1/_*fasta2} >> my_out_file; done
```

This example uses two variables in a for loop. The content of the second variable gets specified in each loop by a replace function.
Runs BLAST2 in all-against-all mode and writes results into one file ('-F F' turns low-complexity filter off).

```
for i in *.fasta; do for j in *.fasta; do blast2 -p blastp -F F -i $i -j $j >> my_out_file; don
e; done;
```

How to write a real shell script

- create file which contains in first line:

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

- place shell commands in file
- run <chmod +x my_shell_script> to make it executable
- run shell script like this:

```
./my_shell_script
```

- when you place it into /usr/local/bin you only type its name from any user account

Remote Copy: wget, scp, ncftp

wget: File Download from the Web

```
wget <ftp://> # file download from www; add option '-r' to download entire directories
```

scp: Secure Copy Between Machines

```
scp source target # Use form 'userid@machine_name' if your local & remote user ids are differen
t. If they are the same, use only 'machine_name'.scp user@remote_host:file.name . # Copies file
from server to local machine (type from local machine prompt). The '.' copies to pwd, you can s
pecify any directory, use wildcards to copy many files.

scp file.name user@remote_host:~/dir/newfile.name # Copies file from local machine to server.

scp -r user@remote_host:directory/ ~/dir  # Copies entire directory from server to local machin
e.
```

Nice FTP

```
ncftp

 ncftp> open ftp.ncbi.nih.gov

 ncftp> cd /blast/executables

 ncftp> get blast.linux.tar.Z (skip extension: @)

 ncftp> bye
```

Archiving and Compressing

Creating Archives

```
tar -cvf my_file.tar mydir/    # Builds tar archive of files or directories. For directories, e
xecute command in parent directory. Don't use absolute path.

tar -czvf my_file.tgz mydir/   # Builds tar archive with compression of files or directories. F
or dirs, execute command in parent directory.

zip -r mydir.zip mydir/        # Command to archive a directory (here mydir) with zip.

tar -jcvf mydir.tar.bz2 mydir/ # Creates *.tar.bz2 archive
```

Viewing Archives

```
tar -tvf my_file.tar

tar -tzvf my_file.tgz
```

Extracting Archives

```
tar -xvf my_file.tar

tar -xzvf my_file.tgz

gunzip my_file.tar.gz # or unzip my_file.zip, uncompress my_file.Z,or bunzip2 for file.tar.bz2

find -name '*.zip' | xargs -n 1 unzip # this command usually works for unzipping many files tha
t were compressed under Windows

tar -jxvf mydir.tar.bz2 # Extracts *.tar.bz2 archive
```

Try also:

```
tar zxf blast.linux.tar.Z
tar xvzf file.tgz
```

Important options:

```
f: use archive file
p: preserve permissions
v: list files processed
x: exclude files listed in FILE
z: filter the archive through gzip
```

Simple Installs

Systems-wide installations

Installations for systems-wide usage are the responsibility of students. If you would like to have something installed, follow the B. and C. movie presentations series.

Environment Variables

```
xhost user@host                # adds X permissions for user on server.
echo $DISPLAY                  # shows current display settings
export DISPLAY=<local_IP>:0    # change environment variable
unsetenv DISPLAY               # removes display variable
env                            # prints all environment variables
```

List of directories that the shell will search when you type a command:

```
echo $PATH
```

You can edit your default DISPLAY setting for your account by adding it to file

```
.bash_profile
```