
NGS Course Nove Hradý Documentation

Release 1.0

Libor Morkovsky, Vaclav Janousek

April 21, 2015

1	Installation instructions	3
1.1	Windows	5
1.2	Mac OS X and Linux	5
1.3	Testing	6
2	Connecting to the virtual machine	7
2.1	Connect to control the machine	7
2.2	Connect to copy files	9
2.3	Conect to RStudio	10
3	Practical part (Unix introduction)	11
3.1	Basic orientation	11
3.2	Installing software	12
3.3	FASTQ	13
3.4	GFF, VCF, BED	16
4	Reference manual for UNIX introduction	21
4.1	Basic orientation in UNIX	21
4.2	Exploring and basic manipulation with data	23
4.3	Building commands	25
4.4	Advanced text manipulation (sed)	27
4.5	More complex data manipulation (awk)	27
5	Important NGS formats	29
5.1	FASTQ - Short reads	29
5.2	SAM - Reads mapped to reference	29
5.3	BED and GFF - Annotations	29
5.4	VCF - Variants in individuals	29
6	Course materials preparation	31
6.1	Online documentation	31
6.2	VirtualBox image	32
6.3	Slide deck	34
7	Best practice	35
7.1	Easiest ways to get UNIX	35
7.2	Essentials	35
7.3	Data organization	35
7.4	Building command lines	36

7.5	Parallelization	37
8	Slide decks	39
9	Other speakers' materials	41
9.1	Petri Kemppainen - LDna	41
9.2	Jean Francois Martin	41

This course aims to introduce the participants to UNIX - an interface that is one of the most convenient options for working with big data. Special attention is put on working with Next Generation Sequencing (NGS) data. Most of the NGS data formats were designed to be textual, and textual data is where UNIX excels in its versatility. With just a combination of basic UNIX tools one can achieve results that would require finding specialized software in other environments, with just few commands.

Not knowing ‘the right way’ to do things can be intimidating for the beginning users, so an additional section stressing the ‘best practice’ was added to the supplementary materials.

Initial instructions:

Installation instructions

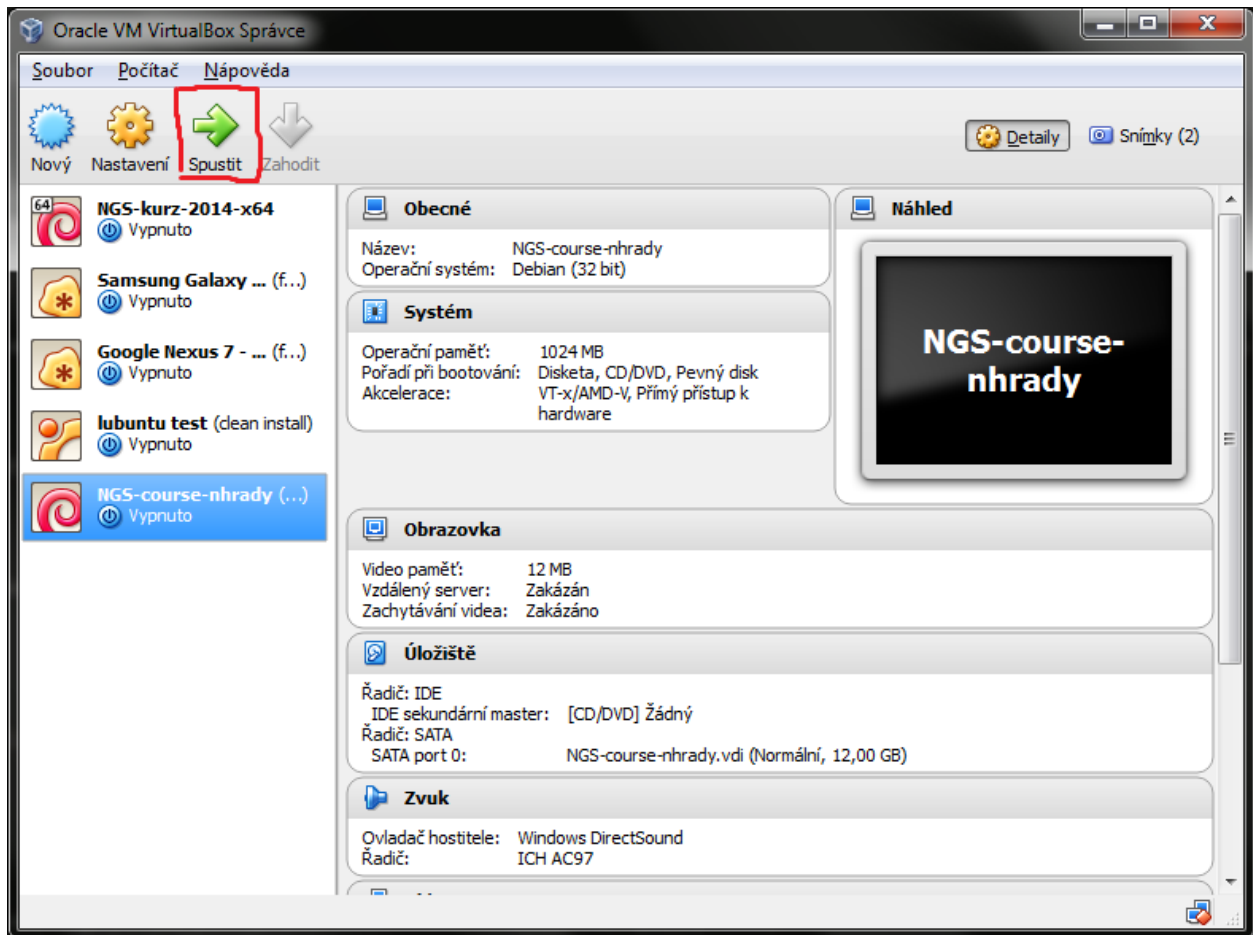
We will be using a virtual computer preinstalled with Debian Linux and sample data necessary for the excersises.

Note: You need to install it even if your main system is Linux / Mac OS X!

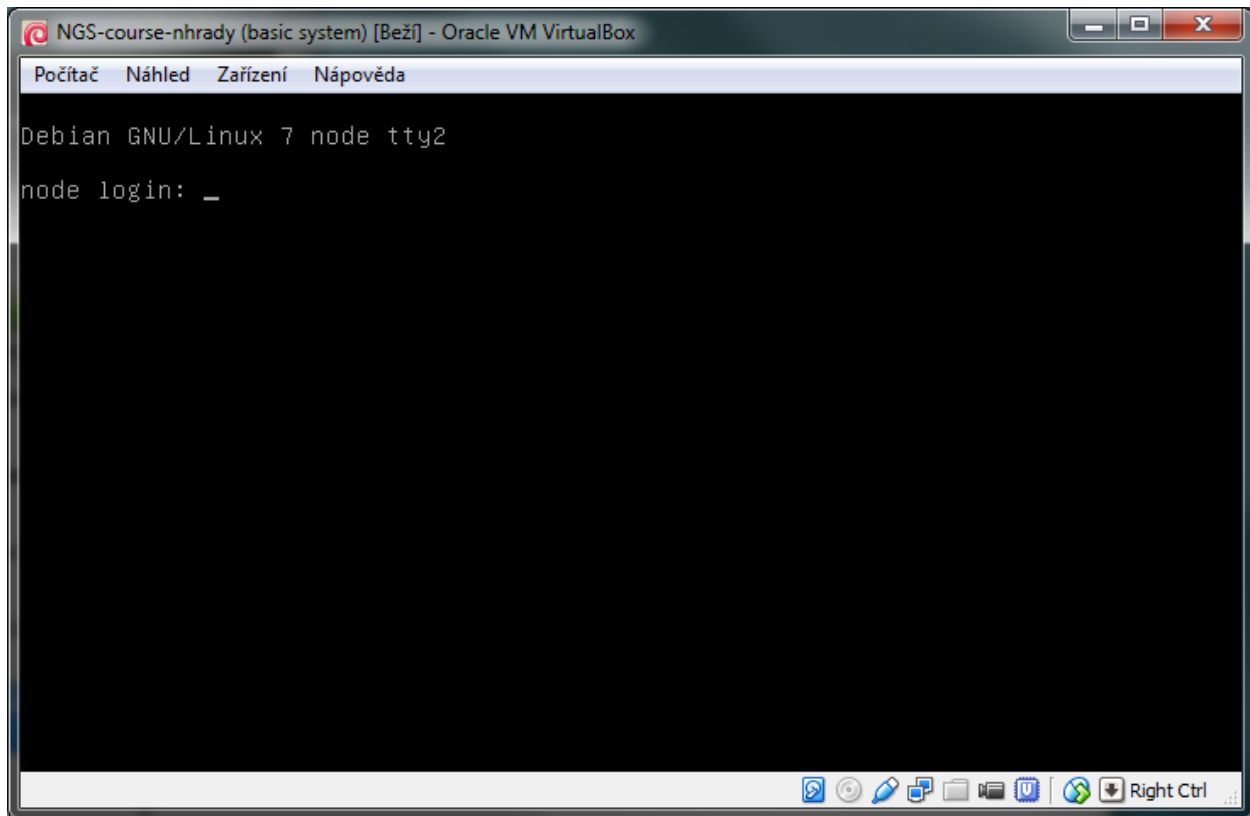
Installation steps (it should take about 10 minutes):

- Install VirtualBox (<https://www.virtualbox.org/wiki/Downloads>). It works on Linux and Mac too.
- Download the virtual machine image from this link: <http://goo.gl/ofOtS9> You'll get a single file with .ova extension on your hard drive.
- You can either double click the .ova file, or run VirtualBox, and choose `File > Import Appliance`. Follow the instructions after the import is started.

After asuccessful instalation you should see something like this (only the machine list will contain jsut one machine). Check whether you can start the virtual machine: click `Start` in the main VirtualBox window:



After a while you should see something like this:



You don't need to type anything into that window, just checking that it looks like the screenshot is enough.

Machine configuration details:

- Administrative user: *root*, password: *debian*
- Normal user: *user*, password: *user*
- ssh on port 2222
- RStudio on port 8787

In case of any problems try to find me (Libor Morkovsky) in Nove Hradky, we'll try to resolve it before the course.

1.1 Windows

Install PuTTY and WinSCP. PuTTY will be used to control the virtual computer. WinSCP will be used to transfer files between your computer and the virtual computer.

- PuTTY (<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html> - look for putty.exe)
- WinSCP (<http://winscp.net/eng/download.php> - look for Installation package).

1.2 Mac OS X and Linux

Ssh is used to control the virtual computer. It should be installed in your computer.

Files can be transferred with `scp`, `rsync` or `lftp` (recommended) from the command line. `Scp` and `rsync` could be already installed in your system, if you want to use `lftp`, you'll probably have to install it yourself.

Mac users that prefer graphical clients can use something like *CyberDuck*. See <http://apple.stackexchange.com/questions/25661/whats-a-good-graphical-sftp-utility-for-os-x>.

1.3 Testing

Try to log in following the instructions in *Connect to control the machine*.

When you're logged in, check your internet connection from the virtual machine. Your main computer has to be connected to the internet, of course. Copy the following command, and paste it to the command prompt (click right mouse button in PuTTY window).

```
wget -q -O - http://goo.gl/n8XK2Y | head -1  
# <!DOCTYPE html>
```

If the `<!DOCTYPE html>` does not appear, something is probably wrong with the connection.

Connecting to the virtual machine

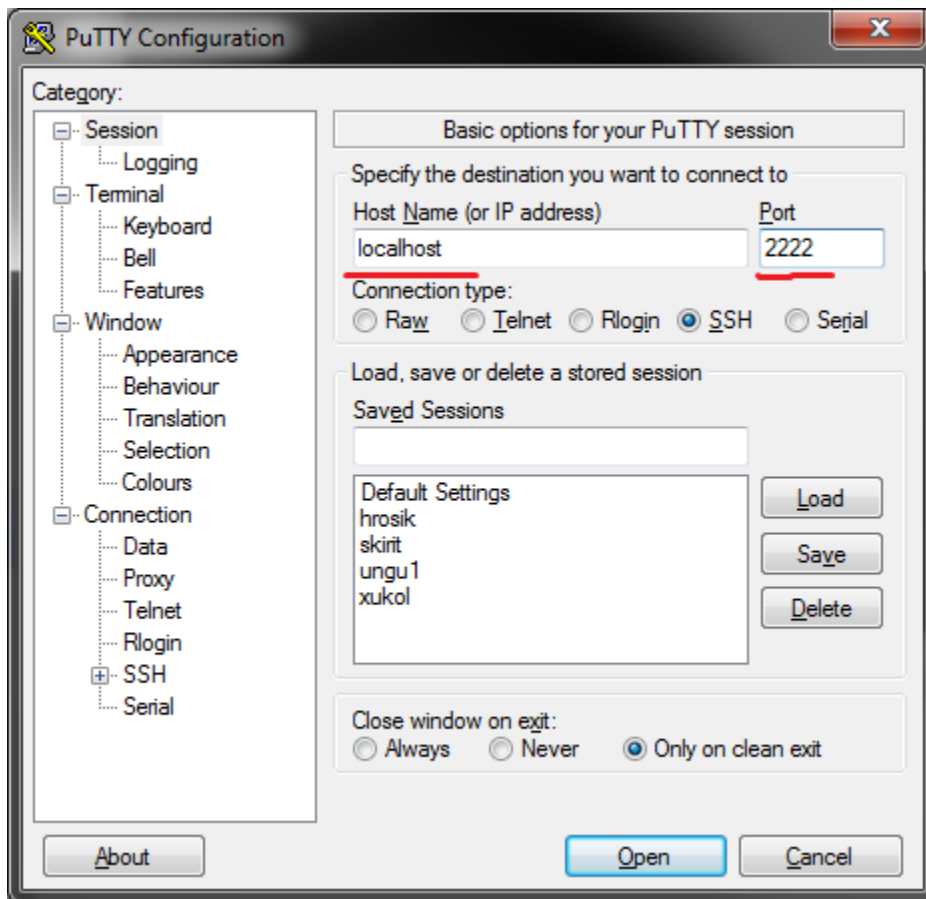
Note: You need to start the virtual machine first!

2.1 Connect to control the machine

To control the machine, you need to connect to the ssh service.

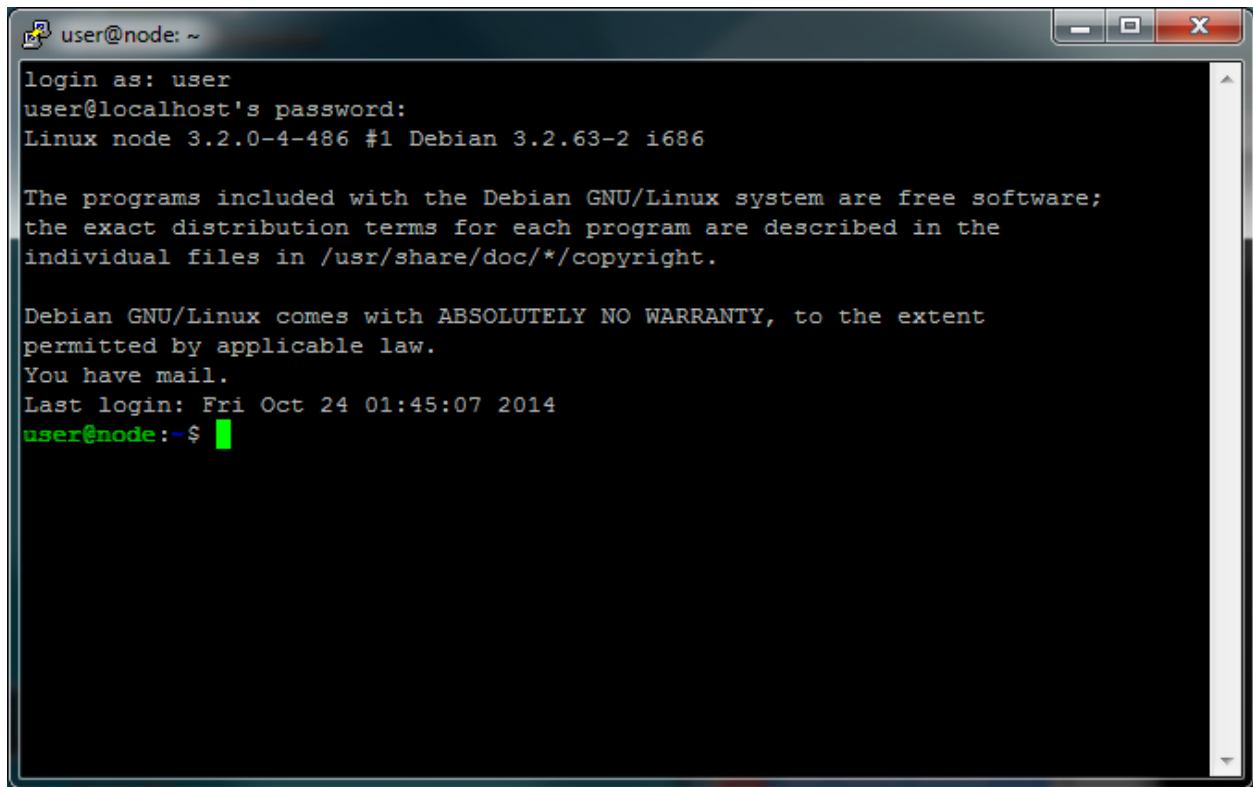
In Windows this is done with PuTTY.

- start PuTTY
- fill Host Name: `localhost`
- fill Port: `2222`
- click Open or press <Enter>



In the black window that appears, type your credentials:

- login as: `user`
- `user@localhost`'s password: `user`

A terminal window titled 'user@node: ~' with standard window controls. The text inside shows a login sequence: 'login as: user', 'user@localhost's password:', and system information 'Linux node 3.2.0-4-486 #1 Debian 3.2.63-2 i686'. It then displays the Debian GNU/Linux license notice, including the warranty disclaimer and the last login time 'Fri Oct 24 01:45:07 2014'. The prompt 'user@node:~\$' is shown at the bottom with a green cursor.

```
login as: user
user@localhost's password:
Linux node 3.2.0-4-486 #1 Debian 3.2.63-2 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

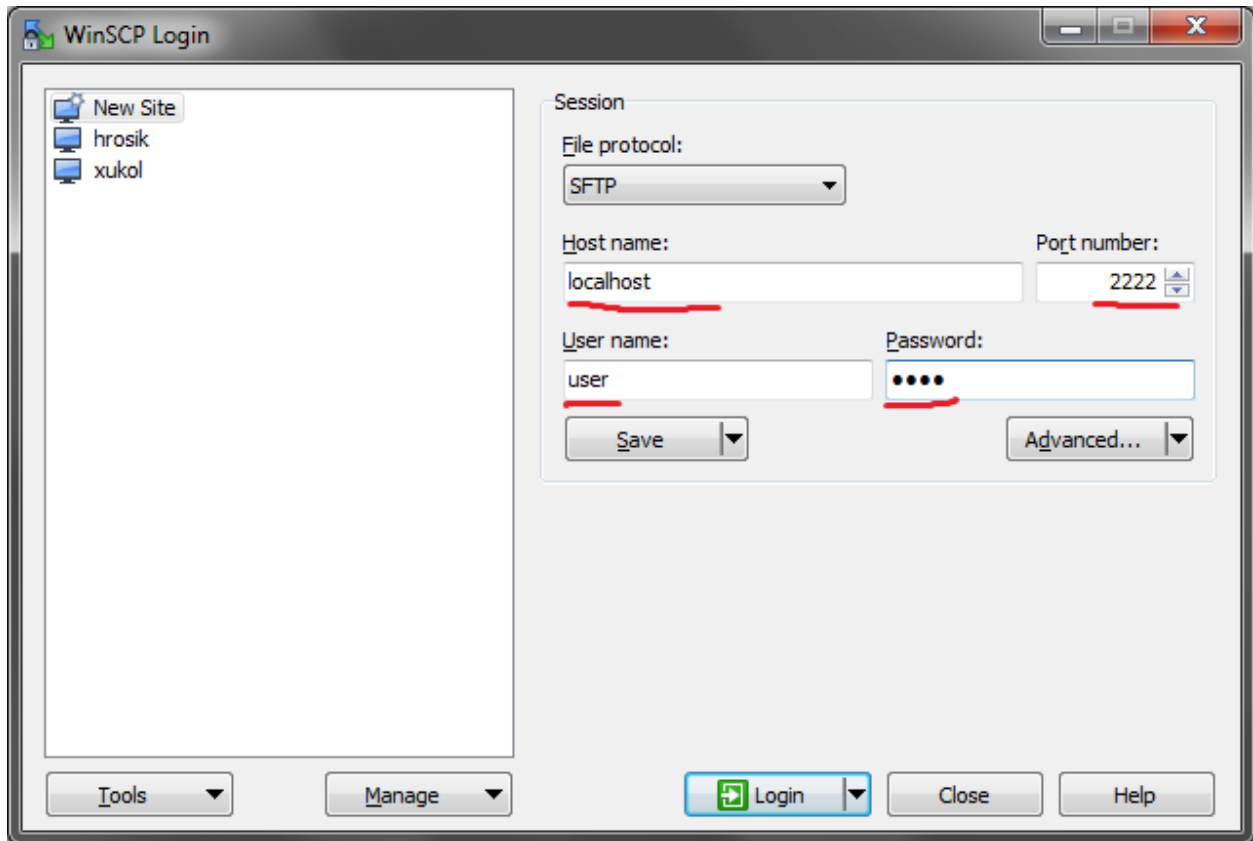
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have mail.
Last login: Fri Oct 24 01:45:07 2014
user@node:~$
```

In Mac OS X or Linux, this is done with ssh tool:

```
ssh -p 2222 user@localhost
```

2.2 Connect to copy files

In Windows, WinSCP is used to copy files to Linux machines. You use the same information as for PuTTY to log in.



In Mac OS X or Linux, the most simple command to copy a file into a home directory of `user` on a running virtual machine is:

```
scp -P 2222 myfile user@localhost:~
```

2.3 Conect to RStudio

This is the easiest one, just click this link: [Open RStudio](#). Login with the same credentials (user, user).

Course contents:

Practical part (Unix introduction)

3.1 Basic orientation

1. Use of multiple windows
2. Use some ‘move around’ commands to explore directory structure
3. Prepare data in your home directory

Note: Before we start, please, install this software:

```
sudo apt-get install htop
```

Those without internet acces in the virtual machine:

- download http://ftp.cz.debian.org/debian/pool/main/h/htop/htop_1.0.1-1_i386.deb
- transfer it to virtual machine (WinSCP, scp)
- run `sudo gdebi htop_1.0.1-1_i386.deb`

1. Use multiple windows

First, type `screen` in your terminal:

```
screen
```

You created first window. To create another one press `ctrl+a c`. To switch between them press `ctrl+a space`.

To see how useful it is you can try to run `htop` in one of these windows

```
htop
```

Htop displays CPU and memory utilization of the (virtual) computer. Continue your work in another one (`ctrl+a space`). You can switch back to `htop` window to monitor progress of some lengthy operation.

2. Use some ‘move around’ commands to explore directory structure

Your bash session has a *working directory*. All paths you use are supposed to start in your working directory, unless you start them with `/`. Try the following command in the order they are provided, and figure out what they do. Then use your knowledge to explore the directory structure of the virtual machine.

```
pwd
ls
ls /
```

```
cd /
pwd
ls
cd
pwd
```

In case you don't know ask or go to *Moving around & manipulation with files and directories*.

3. Prepare data in your home directory

All data we are going to use today are located in directory `/data` and as that they are potentially available to all users. However, we want to have them in our own data directory. Without a need to copy them all to our directory we can have them there using symbolic link (`ln -s`). This keeps data in their original location but creates reference to them.

In case you did not know where the data are but you knew their names or suffixes you could use `locate` command. We know they have `fastq`, `vcf` and `gff3` suffixes:

```
locate fastq
locate vcf
locate gff3
```

Once we know their actual position we can create symbolic links:

```
mkdir data # create directory data
cd data # go to your new data directory
ln -s /data/00-reads 00-reads
ln -s /data/01-genome 01-genome
ln -s /data/02-variants 02-variants
```

We created symbolic links to three places: `00-reads`, `01-genome` and `02-variants`. We can check them by typing:

```
ls -l
```

3.2 Installing software

The easiest way to install software is via a package manager as you've seen in the beginning (`apt-get` for all Debian variants). When the required software is not in the repositories, or one needs the latest version, it's necessary to take the more difficult path. The canonical UNIX way is:

```
wget -O - ..url.. | tar xvz # download the 'tarball' from internet
cd ..unpacked directory.. # set working directory to the project directory
./configure # check your system and choose the way to build it
make && sudo make install # convert source code to machine code and if successful, copy the result
```

In our example, some steps are omitted. We'll install `bedtools` program from a github repository. User installed software can be found in `~/sw` directory. To install a new software go to this directory:

```
cd ~/sw
```

When the software source code is in a single file (*tarball*), `wget` command is the best option to get the file. The latest versions are usually not packaged, and many of the tools can be found at GitHub. To get stuff from GitHub, `git clone` command is usually the easiest option.

```
git clone https://github.com/arq5x/bedtools2
```


For those without internet access in their virtual machines - you need to download the content to your normal computer and then transfer it to the virtual machine.

- download <https://github.com/arq5x/bedtools2/archive/master.zip>
- transfer it to the virtual machine with WinSCP (Windows) or scp (Mac or Linux)
- unpack the file with `unzip`
- rename the folder with `mv` to `bedtools2`

This creates a *clone* of the online repository in directory `bedtools2`.

```
cd bedtools2
```

To compile (convert from text source form to machine executable form) software on UNIX use the `make` command:

```
make
```

It should take a while, so you can flip to your *htop* window with `ctrl-a space` and watch the CPU spin;)

When `bedtools` is compiled you have to copy `bedtools` binaries to `/usr/local/bin` directory for UNIX system to find the program when calling from any place in the system.

Warning: Before you use command below to copy binaries make sure you are really in directory you want to be!

```
cd bedtools2/bin
ls # Check that you are really in directory you want to be!
sudo cp * /usr/local/bin
```

We used two commands: `sudo` and `cp`. The `sudo` command tells the system that we want to make changes in system directories and as such we are asked for password. This step prevents us from harming system. The `cp` command copies all `bedtools` binaries from local `bin` directory to the system binary repository.

Note: We used the `*` symbol which tells the system that all files in the current directory should be selected. We explain this later.

3.3 FASTQ

Explore lengths of short reads in FASTQ files:

1. Explore FASTQ files
2. Find out how many reads are there in each file and in total
3. Calculate summary statistics of read lengths
4. Find primers in FASTQ files

1. Explore FASTQ files

To view contents of FASTQ file go to `data/00-reads` directory see contents of the directory using `ls` command and view file using `less`:

```
cd data
ls
less -SN 00-reads/GS60IET02.RL1.fastq
```

Note: You don't have to type the whole file name. Try to use TAB completion!

Try use and unuse `-S` and `-N` options and see what's the difference.

You can also use `head` command to view first lines:

```
head 00-reads/GS60IET02.RL1.fastq      # the default is to show 10 lines
head -n 20 00-reads/GS60IET02.RL1.fastq # to show first 20 lines, use -n 20
```

or `tail` command to view last 20 lines:

```
tail -n 20 00-reads/GS60IET02.RL1.fastq
```

2. How many reads are there?

We found out that FASTQ files has certain structure and that each read takes four lines (ID, sequence, quality, +; see NGS formats). To obtain number of sequences in each FASTQ file we build a pipeline by combination of `grep` and `wc` commands along with UNIX feature called *globbing*.

First, let's try to see what's globbing:

```
echo 00-reads/*.fastq
```

When bash encounters a special character like `*` or `?`, it tries to match filename patterns in the directory structure, where `?` match for any single character and `*` for 0 or more any characters, respectively. It can, however, match more complex patterns.

So let's get back to counts of reads...

We can first try to get counts of *lines* in each file simply by typing:

```
wc -l 00-reads/*.fastq
```

However, to obtain counts of *reads* in each file we have to select just ID lines using `grep` command:

```
grep "^@[0-9A-Z]*$" 00-reads/*.fastq | wc -l
```

Command `grep` enables to search file for specific character or string of characters. Here, we used so-called regular expressions to specify the pattern `grep` is supposed to search for. Regular expressions is a very concise and 'magical' way to describe text patterns. Let's go through our expression piece by piece.

- `^` marks beginning of line - otherwise `grep` would search anywhere in the line
- the square brackets `[]` represent a character of given class (0 to 9 or A to Z)
- the `*` is a count suffix for the square brackets, saying there should be zero or more of such characters
- `$` marks end of line - that means the whole line has to match the pattern

If you like regular expressions, you can hone your skills at <https://regex.alf.nu/>.

3. Calculate summary statistics of read lengths

In this particular task we will need first two lines (ID, sequence) of FASTQ files for each sequence. The simplest way to do that is to use UNIX built-in programmatic interface called `awk`. This program enables to efficiently handle the data of various complexity. We build a bit more complex pipeline which is going to combine `awk` tool along with other commands (`tr`, `tail`, `tabtk`).

This is how the whole pipeline looks like:

```
awk '{ if( (NR+3) % 4 == 0 || (NR+2) % 4 == 0 ){print $0} }' 00-reads/*.fastq |
tr '\n@' '\t\n' |
tail -n +2 |
awk -F '$\t' 'BEGIN{OFS=FS}{ print $1,length($2)}' |
tabtk num -c 2
```

Now we can go step by step through the proces of building it (this is how we did it, there's no other magic):

In the first step we are going to send all FASTQ files to command written in `awk`. This command is supposed to return just ID and sequence for each read (i.e first and second line).

```
awk '{ if( (NR+3) % 4 == 0 || (NR+2) % 4 == 0 ){print $0} }' 00-reads/*.fastq | less -S
```

`NR` is an `awk` built-in variable set to the number of current line (see reference for others built-in variables).

Now, we created a file with read IDs and sequences where these two alter by line. To create a file with IDs and sequences on the same line we take advantage of the structure of the file. We use `tr` command which replaces and deletes characters in file.

```
awk '{ if( (NR+3) % 4 == 0 || (NR+2) % 4 == 0 ){print $0} }' 00-reads/*.fastq |
tr '\n@' '\t\n' |
tail -n +2 |
head
```

First we replace symbol for newlines (`\n`) with symbol with TAB (`\t`). This concatenates all lines into one, each one separated by TAB. Second, we want to have record for each read (i.e. ID, sequence) in one line. Thus, we introduce newline symbol (`\n`) instead of `@` symbol. Lastly, as we find out that first line is empty, we remove it by invoking `tail` command. This command with `-n +2` option takes all lines throughout the file starting at line two.

Now, we have TAB delimited file with two columns. The first one is for read ID and second one is the read sequence. However, we are interested in the length of sequence. So we use `awk` again to calculate the length for each read:

```
awk '{ if( (NR+3) % 4 == 0 || (NR+2) % 4 == 0 ){print $0} }' 00-reads/*.fastq |
tr '\n@' '\t\n' |
tail -n +2 |
awk -F '$\t' 'BEGIN{OFS=FS}{ print $1,length($2)}' |
head
```

The syntax of this command is simple. First, we need to set TAB as separator because by default `awk` considers white space as separator. To set TAB as input and output field separator we use two other built-in variables (`FS`, `OFS`). The input field separator (`FS`) is set by `-F` option. The output field separator is set in the `BEGIN{ }` part by passing value of `FS` to `OFS`. Next, in the middle section we print for each line (i.e. each read) the first column (read ID) and length of sequence. The length of sequence is obtained using `awk` built-in function `length()`. The `$$\t` is a way how to pass TAB character - because if you just press it on the keyboard, it invokes bash autocompletion and does not type the character.

Lastly, we calculate read length summary statistics using program `tabtk` we installed at the beginning:

```
awk '{ if( (NR+3) % 4 == 0 || (NR+2) % 4 == 0 ){print $0} }' 00-reads/*.fastq |
tr '\n@' '\t\n' |
tail -n +2 |
awk -F '$\t' 'BEGIN{OFS=FS}{ print $1,length($2)}' |
tabtk num -c 2
```

3.1 Alternatives (optional)

Finally to show you that you can find less concise, but also less understandable alternatives in UNIX, the first awk command can be shortened to almost a half, but you have to know a special rule saying that the conditions can precede the actions, and that the default action is `print $0;`.

```
awk 'NR % 4 == 1 || NR % 4 == 2' 00-reads/*.fastq |
  tr '\n@' '\t\n' |
  tail -n +2 |
  awk -F '$'\t' 'BEGIN{OFS=FS}{ print $1,length($2)}' |
  tabtk num -c 2
```

Or another tool (`sed`) can be used to get even shorter command (the original was 60 characters, this one is 22). But when trying to get the shortest possible code, one has to keep in mind that he's writing the code once, but will be reading/reusing/fixing it several times. Good readability and understandability is always the most important criterion. You can also use some knowledge about the data to shorten your commands - you're sure here, that there are no whitespace characters in the sequence names and sequences themselves, so you can stick with the default field separator.

```
sed -n -e 1~4p -e 2~4p 00-reads/*.fastq |
  tail -n +2 |
  tr '\n@' '\n' |
  awk '{print $1,length($2)}' |
  tabtk num -c 2
```

```
# finally, if you don't care about the sequence names
sed -n 2~4p 00-reads/*.fastq | awk '{print length}' | tabtk num
```

4. Find primers in FASTQ files

Reads in FASTQ files contain adaptors that were used for reverse transcription of the mRNA. Try to identify them and visualize them using basic UNIX commands.

First, we store the primer sequences into shell variables which we use later. This steps help us to get families to what it is and how to work with shell variables in UNIX environment.

To set primer sequences into `PRIMER#` variable type:

```
PRIMER1="AAGCAGTGGTATCAACGCAGAGTACGCGGG"
PRIMER2="AAGCAGTGGTATCAACGCAGAGT"
```

To interpret a string as shell variable name, prefix it with `$`:

```
echo $PRIMER1
# AAGCAGTGGTATCAACGCAGAGT
```

The `echo` command printed contents of the variable. However, the variable can be used in any other command in UNIX. We use them in searching for primers in FASTQ files:

```
grep --color=always $PRIMER1 00-reads/*.fastq | less -RS
```

Here, the `grep`'s coloured output was sent to `less` which kept the colors of the matched primers. To colour matches add `--color=always` in `grep` command and `-R` option in `less`.

3.4 GFF, VCF, BED

Find SNPs and INDELs identified using reads which overlap with 5' untranslated regions.

1. Explore GFF file
2. Create BED file for 5' untranslated regions

3. Explore VCF files
4. Create BED file for SNPs and INDELs
5. Join the two BED files using BEDTools

1. Explore GFF file (less)

2. Create BED file for 5' untranslated regions

We're creating a new type of result from genome annotation data. It is wise to create a new directory to contain this type of analysis on the genome:

```
mkdir 03-utr-analysis
```

The whole command looks like this:

```
<01-genome/luscinia_small.gff3 grep 5utr |
tr ';' ' ' |
sed 's/Name=/' |
awk -F '$'\t' 'BEGIN{OFS=FS}{print $1,$4-1,$5,$10}' \
>03-utr-analysis/utrs.bed
```

Let's go step by step:

First, we need to filter out records corresponding to 5' UTRs in the GFF file. For this task we can use `grep` function and `less` to see the results:

```
<01-genome/luscinia_small.gff3 grep 5utr | less -S
```

Having just 5' UTR records we need to remove and resort some columns. We use combination of `sed`, `tr` and `awk` commands:

```
<01-genome/luscinia_small.gff3 grep 5utr |
tr ';' ' ' |
sed 's/Name=/' |
less -S
```

First, we use `tr` command to extract gene ID. We replace semicolon and white space by TAB separator. These replacements cause the INFO column to split into three. Subsequently we delete 'Name=' part in the gene ID column using `sed` command.

```
<01-genome/luscinia_small.gff3 grep 5utr |
tr ';' ' ' |
sed 's/Name=/' |
awk -F '$'\t' 'BEGIN{OFS=FS}{print $1,$4-1,$5,$10}' |
less -S
```

Further, as BEDTools assume zero based coordinate system, we use `awk` to subtract one from all start coordinates.

We can redirect the whole output into `utrs.bed` file in our new analysis directory:

```
<01-genome/luscinia_small.gff3 grep 5utr |
tr ';' ' ' |
sed 's/Name=/' |
awk -F '$'\t' 'BEGIN{OFS=FS}{print $1,$4-1,$5,$10}' \
>03-utr-analysis/utrs.bed
```

3. Explore VCF file (less)

4. Create BED file out of VCF file for SNPs and INDELs

```
grep -hv ^# 02-variants/*.vcf |
awk -F $'\t' '
BEGIN{OFS=FS}
{ if(length($4)==1)
  { print $1, ($2-1), ($2+length($4)-1), "SNP" }
  else
  { print $1, ($2-1), ($2+length($4)-1), "INDEL" }
}' \
>02-variants/variants.bed
```

The first line can be improved with *pv*, which gives a progress indicator.

```
pv 02-variants/*.vcf |
grep -v ^# |
awk -F $'\t' '
BEGIN{OFS=FS}
{ if(length($4)==1)
  { print $1, ($2-1), ($2+length($4)-1), "SNP" }
  else
  { print $1, ($2-1), ($2+length($4)-1), "INDEL" }
}' \
>02-variants/variants.bed
```

First, we use inverted *grep* command (*-v* option) to remove INFO lines (beginning with # symbol). Also, as we *grep* from multiple files (i.e. * globbing) we use option *-h* to suppress file names in the output. Try run *grep* with and without *-h* option:

```
grep -hv ^# 02-variants/*.vcf | head
```

Second, we want to distinguish between SNPs and INDELs and create BED file. The difference is in length of REF column in VCF files. SNPs have always only single character, whereas INDELs have always at least two. So we can use easy *if()* condition in *awk* based on length of REF column. Also, as in the VCF file is only first position of the variant, when creating BED file one has to calculate the second coordinate. So the start position of a SNP is one minus the actual position, whereas the end position is the actual position:

```
grep -hv ^# 02-variants/*.vcf |
awk -F $'\t' '
BEGIN{OFS=FS}
{ if(length($4)==1)
  { print $1, ($2-1), ($2+length($4)-1), "SNP" }
  else
  { print $1, ($2-1), ($2+length($4)-1), "INDEL" }
}' |
head
```

Finally, the output can be redirected into *variants.bed*.

5. Join the two BED files using BEDTools

Finally, we are interested in how many of SNPs and INDELs are located in 5' UTRs. For this task we use BEDTools that represent a suite of tools to do easily so-called “genome arithmetic”.

Full pipeline:

```
bedtools intersect -a 01-genome/utrs.bed -b 02-variants/variants.bed -wa -wb |
cut -f 4,8 |
sort -k2,2 |
bedtools groupby -g 2 -c 1 -o count
```

First, we use BEDTools tool `intersect` to find an overlap between SNPs, INDELs and 5' UTRs.

```
bedtools intersect -a 01-genome/utrs.bed -b 02-variants/variants.bed -wa -wb | head
```

Here, the `-a` and `-b` options state for file a and file b. Also, it is necessary to specify which of the two files (or both of them) to print in the output (`-wa`, `-wb`).

As you may notice, the output contains eight columns (i.e. four for each file). For us, however, what is important is only information on gene ID and type of variant (SNPs or INDELs). So we cut out only these two columns using `cut` command:

```
bedtools intersect -a 01-genome/utrs.bed -b 02-variants/variants.bed -wa -wb |
  cut -f 4,8 |
  head
```

The `-f` option in the `cut` command states for specification of columns which are supposed to be cut out.

Now, we want to obtain counts of SNPs and INDELs overlapping with 5' UTRs. We use another BEDTools tool - `groupby`. This tool enables to group data based on column of choice and to do some summary statistics on another another one. Before grouping, however, we need to sort the data according to the column which we use as a grouping column:

```
bedtools intersect -a 01-genome/utrs.bed -b 02-variants/variants.bed -wa -wb |
  cut -f 4,8 |
  sort -k2,2 |
  bedtools groupby -g 2 -c 1 -o count
```

To sort based on certain column one has to use `-k` option along with specification of range (in columns) of sorting. If we want to sort based on one column - as in the case above - we specify range using column position. Here, we sort based on second column so we specify range as `-k2,2`. The BEDTools tool `groupby` has several options. `-g` option specifies column based on which we group, `-c` option specifies column to which we apply summary statistics and `-o` option specifies type of summary statistics (see manual at <http://bedtools.readthedocs.org>).

Reference manual for UNIX introduction

4.1 Basic orientation in UNIX

Multiple windows (screen)

You're all used to work with multiple windows (in MS Windows;). You can have them in UNIX as well. The main benefit, however, is that you can log off and your programs keep running.

To go into a screen mode type:

```
screen
```

Once in screen you can control screen itself after you press the master key (and then a command): `ctrl+a` key. To create a new window within the screen mode, press `ctrl+a c` (create). To flip among your windows press `ctrl+a space` (you flip windows often, it's the biggest key available). To detach screen (i.e. keep your programs running and go home), press `ctrl+a d` (detach).

To open a detached screen type:

```
screen -r # -r means restore
```

To list running screens, type:

```
screen -ls
```

Controlling processes (htop/top)

`htop` or `top` serve to see actual resource burden for each running process. `Htop` is much nicer variant of standard `top`. You can sort the processes by memory usage, CPU usage and few other things.

Getting help (man)

Just any time you're not sure about program option while building a command line, just flip to next screen window (you're always using screen for serious work), and type `man` and name of the command you want to know more about.

```
man screen
```

4.1.1 Moving around & manipulation with files and directories

Basic commands to move around and manipulate files/directories.

```
pwd    # prints current directory path
cd     # changes current directory path
ls     # lists current directory contents
ll     # lists detailed contents of current directory
mkdir  # creates a directory
rm     # removes a file
rm -r  # removes a directory
cp     # copies a file/directory
mv     # moves a file/directory
locate # tries to find a file by name
```

Usage:

cd

To change into a specific subdirectory, and make it our present working directory:

```
cd go/into/specific/subdirectory
```

To change to parent directory:

```
cd ..
```

To change to home directory:

```
cd
```

To go up one level to the parent directory then down into the directory2:

```
cd ../directory2
```

To go up two levels:

```
cd ../../
```

ls

To list all (including hidden) files and directories (-a) in current in given folder along with human readable (-h) size of files (-s), type:

```
ls -ash
```

mv

To move a file data.fastq from current working directory to directory /home/directory/fastq_files, type:

```
mv data.fastq /home/directory/fastq_files/data.fastq
```

cp

To copy a file data.fastq from current working directory to directory /home/directory/fastq_files, type:

```
cp data.fastq /home/directory/fastq_files/data.fastq
```

locate

This command enables to find any string on system. It helps find location of given files.

So to locate file data.fastq type:

```
locate data.fastq
```

This command uses database of files and directories which updates just once a day. When you look for recent files you may not find them. So to search for these files one has to update database before:

```
sudo updatedb
```

Symbolic links

Symbolic links refer by their names to some files or directories in different location. It is useful when one wants to work with some general files accessible to more users but at the same time to have them in local directory. Also, it is useful when one works at multiple projects and uses same files (especially large ones). Instead of copying them into each project directory one can use simply symbolic links.

Symbolic link can be created by:

```
ln -s /data/genomes/luscinia/genome.fa genome/genome.fasta
```

This command creates symbolic link on file in general location (/data/genomes/luscinia/genome.fa) and the link is created in subdirectory to the current working directory (genome/genome.fasta).

4.2 Exploring and basic manipulation with data

less

Program to view (but not to change) and navigate throughout the contents of text files. As it reads only part of a file on the screen (i.e. does not have to read entire file before starting), it has fast load times for large files.

To view text file while disabling line wrap and add line numbers add options `-S` and `-N`, respectively.

```
less -SN data.fasta
```

To navigate within the text file while viewing use:

Key	Command
Space bar	Next page
b	Previous page
Enter key	Next line
/<string>	Look for string
<n>G	Go to line <n>
h	Help
q	Quit

cat

Utility which outputs the contents of a specific file and can be used to concatenate and list files.

```
cat seq1_a.fasta seq1_b.fasta > seq1.fasta
```

head

By default, this utility prints first 10 lines. The number of first n lines can be specified by `-n` option.

To print first 50 lines type:

```
head -n 50 data.txt
```

tail

By default, this utility prints last 10 lines. The number of last *n* lines can be specified by `-n` option as in case of `head`.

To print last 20 lines type:

```
tail -n 20 data.txt
```

To skip first few lines in the file (e.g. to remove header line of the file):

```
tail -n +2 data.txt
```

grep

This utility enables you to search text file(s) for lines matching text patterns. To match given pattern it uses either specific string or regular expressions. Regular expressions enable for a more generic pattern rather than a fixed string (e. g. search for `a` followed by 4 numbers followed by any capital letter - `a[0-9]{4}[A-Z]`).

To obtain one file with list of sequence IDs in multiple fasta files type:

```
grep '>' *.fasta > seq_ids.txt
```

To print all but `#`-starting lines from the vcf file use option `-v` (invert-match):

```
grep -v ^# snps.vcf > snps.tab
```

The `^` mark specifies beginning of line (i.e. it skips all `#` which are not at the beginning of line).

wc

This utility generates set of statistics on either standard input or list of text files. It provides these statistics:

- line count (`-l`)
- word count (`-w`)
- character count (`-m`)
- byte count (`-c`)
- length of the longest line (`-L`)

If specific word provided it returns count of this word in a given file.

To obtain number of files in a given directory type:

```
ls | wc -l
```

The `|` symbol is explained in further section.

cut

Cut out specific columns (fields/bytes) out of a file. By default, fields are separated by TAB. Otherwise, change delimiter using `-d` option. To select specific fields out of a file use `-f` option (position of selected fields/columns separated by commas). If needed to complement selected fields (i.e. keep all but selected fields) use `--complement` option.

Out of large matrix select all but first column and row representing IDs of rows and columns, respectively:

```
< matrix1.txt tail -n +2 | cut --complement -f 1 > matrix2.txt
```

sort

This utility sorts a file based on whole lines or selected columns. To sort numerically use `-n` option. Range of columns used as sorting criterion is specified by `-k` option.

Extract list of SNPs with their IDs and coordinates in genome from vcf file and sort them based on chromosome and physical position:

```
< snps.vcf grep ^# | cut -f 1-4 | sort -n -k2,2 -k3,3 > snps.tab
```

uniq

This utility takes sorted lists and provides unique records and also counts of non-unique records (`-c`). To have more numerous records on top of output use `-r` option for `sort` command.

Find out count of SNPs on each chromosome:

```
< snps.vcf grep ^# | cut -f 2 | sort | uniq -c > chromosomes.tab
```

tr

Replaces or removes specific sets of characters within files.

To replace a characters a and b in the entire file for characters c and d type:

```
tr 'ab' 'cd' < file1.txt > file2.txt
```

Multiple consecutive occurrences of specific character can be replaced by single character using `-s` option. To remove empty lines type:

```
tr -s '\n' < file1.txt > file2.txt
```

To replace lower case to upper case in fasta sequence type:

```
tr "[:lower:]" "[:upper:]" < file1.txt > file2.txt
```

4.3 Building commands

Globbering

Refers to manipulating (searching/listing/etc.) files based on pattern matching using specific characters.

Example:

```
ls
# a.bed b.bed seq1_a.fasta seq1_b.fasta seq2_a.fasta seq2_b.fasta
ls *.fasta
# seq1_a.fasta seq1_b.fasta seq2_a.fasta seq2_b.fasta
```

Character `*` in previous example replaces any number of any characters and it indicates to `ls` command to list any file ending with `".fasta"`.

However, if we look for `fastq` instead, we got no result:

```
ls *.fastq
#
```

Character `?` in following example replaces just right the one character (a/b) and it indicates to `ls` functions to list files containing `seq2_` at the beginning, any single character in the middle (a/b) and ending with `".fasta"`

```
ls
# a.bed b.bed seq1_a.fasta seq1_b.fasta seq2_a.fasta seq2_b.fasta
ls seq2_?.fasta
# seq2_a.fasta seq2_b.fasta

ls
# a.bed b.bed seq1_a.fasta seq1_b.fasta seq2_a.fasta seq2_b.fasta
ls seq2_[ab].fasta
# seq2_a.fasta seq2_b.fasta
```

One can specifically list altering characters (a,b) using brackets []. One may also be more general and list all files having any alphabetical character [a-z] or any numerical character [0-9]:

```
ls
# a.bed b.bed seq1_a.fasta seq1_b.fasta seq2_a.fasta seq2_b.fasta
ls seq[0-9]_[a-z].fasta
# seq1_a.fasta seq1_b.fasta seq2_a.fasta seq2_b.fasta
```

TAB completion

Using key TAB one can finish unique file names or paths without having to fully type them. (try and see)

From this perspective it is important to think about names for directories in advance as it can spare you a lot time in future. For instance, when processing data with multiple steps one can use numbers at beginnings of names:

- 00-beginning
- 01-first-processing
- 02-second-processsing
- ...

Variables

UNIX environment enables to use shell variables. To set primer sequence 'GATACGCTACGTGC' to variable PRIMER1 in a command line and print it on screen using echo, type:

```
PRIMER1=GATACGCTACGTGC
echo $PRIMER1
# GATACGCTACGTGC
```

Note: It is good habit in UNIX to use capitalized names for variables: PRIMER1 not primer1.

Pipes

UNIX environment enables to chain commands using pipe symbol |. Standard output of the first command serves as standard input of the second one, and so on.

```
ls | head -n 5
```

Subshell

Subshell enables to run two commands and capture the output into single file. It can be helpful in dealing with data files headers. Use of subshell enables to remove header, run the set of operations on the data, and later insert the header back to file. The basic syntax is:

```
(command1 file1.txt && command2 file1.txt) > file2.txt
```

To sort data file based on two columns without including header type:

```
(head -n 1 file1.txt && tail -n +2 file1.txt | sort -n -k1,1 -k2,2) > file2.txt
```

Subshell can be used also to preprocess multiple inputs on the fly (saving useless intermediate files):

```
paste <(< file1.txt tr ' ' '\t') <(<file2.txt tr ' ' '\t') > file3.txt
```

4.4 Advanced text manipulation (sed)

sed “stream editor” allows you to change file line by line. You can substitute text, you can drop lines, you can transform text... but the syntax can be quite opaque if you’re doing anything more than substituting *foo* with *bar* in every line (sed 's/foo/bar/g').

4.5 More complex data manipulation (awk)

awk enables to manipulate text data in a very complex way. In fact, it is a simple programming language with functionality similar to regular programming languages. As such it enables enormous variability in ways of how to process text data.

It can be used to write a short script and which can be chained along with UNIX commands in one pipeline. The biggest power of *awk* is that it’s line oriented and saves you lot of boilerplate code that you would have to write in other languages, if you need moderately complex processing of text files. The basic structure of the script is divided into three parts and any of these three parts may or may not be included in the script (according to the intention of user). The first part 'BEGIN{ }' conducts operation before going through the input file, the middle part '{ }' goes throughout the input file and conducts operations on each line separately. The last part 'END{ }' conducts operation after going through the input file.

The basic syntax:

```
< data.txt awk 'BEGIN{<before data processing>} {<process each line>} END{<after all lines are p
```

Built-in variables

awk has several built-in variables which can be used to track and process data without having to program specific feature.

The basic four built-in variables:

- FS - input field separator
- OFS - output field separator
- NR - record (line) number
- NF - number of fields in record (in line)

There is even more built-in variables that we won’t discuss here: RS, ORS, FILENAME, FNR

Use of built-in variables:

awk splits each line into columns based on white space. When a different delimiter (e.g. TAB) is to be used, it can be specified using -F option. If you want to keep this custom Field Separator in the output, you have to set the Output Field Separator as well (there’s no command line option for OFS):

```
< data.txt awk -F '$'\t' 'BEGIN{OFS=FS}{print $1,$2}' > output.txt
```

This command takes file data.txt, extract first two TAB delimited columns of the input file and print them TAB delimited into the output file output.txt. When we look more closely on the syntax we see that the

TAB delimiter was set using `-F` option. This option corresponds to the `FS` built-in variable. As we want TAB delimited columns in the output file we pass `FS` to `OFS` (i.e. output field separator) in the `BEGIN` section. Further, in the middle section we print out first two columns which can be extracted by numbers with `$` symbol (`$1`, `$2`). The numbers correspond to position of the column in the input file. We could, of course, use for this operation the `tr` command which is even simpler. However, the `awk` enables to conduct any other operation on given data.

Note: The complete input line is stored in `$0`.

The `NR` built-in variable can be used to capture each second line in a file type:

```
< data.txt awk '{ if(NR % 2 == 0){ print $0 } }' > output.txt
```

The `%` symbol represents modulo operator which returns the remainder of division. The `if()` condition is used to decide on whether the modulo is 0 or not.

Here is a bit more complex example of how to use `awk`. We write a command which retrieves coordinates of introns from coordinates of exons.

Example of input file:

GeneID	Chromosome	Exon_Start	Exon_End
ENSG00000139618	chr13	32315474	32315667
ENSG00000139618	chr13	32316422	32316527
ENSG00000139618	chr13	32319077	32319325
ENSG00000139618	chr13	32325076	32325184
...

The command is going to be as follows:

When we look at the command step by step we first remove header and sort data based on `GeneID` and `Exon_Start` columns:

```
< exons.txt tail -n +2 | sort -k1,1 -k3,3n | ...
```

Further, we write a short script using `awk` to obtain coordinates of introns:

```
... | awk -F '\t' 'BEGIN{OFS=FS}{
    if(NR==1){
        x=$1; end1=$4+1;
    }else{
        if(x==$1){
            print $1,$2,end1,$3-1; end1=$4+1;
        }else{
            x=$1; end1=$4+1;
        }
    }
}' > introns.txt
```

In the `BEGIN{}` part we set TAB as output field separator. Further, using `NR==1` test we set `GeneID` for first line into `x` variable and intron start into `end1` variable. Otherwise we do nothing. For others records `NR > 1` condition `x==$1` test if we are still within the same gene. If so we print exon end from previous line (`end1`) as intron start and exon start of current line we use as intron end. Next, we set new intron start (i.e. exon end from current line) into `end1`. If we have already moved into new one `x<>$1`) we repeat procedure for the first line and print nothing waiting for next line.

Important NGS formats

A selection of the most commonly used formats in NSG data processing pipelines.

5.1 FASTQ - Short reads

Sequencing instruments produce not only base calls, but usually can assign some quality score to each called base. Fastq contains multiple sequences, and each sequence is paired with quality scores for each base. The quality scores are encoded in text form.

- <http://maq.sourceforge.net/fastq.shtml>

5.2 SAM - Reads mapped to reference

SAM stands for Sequence Alignment/Mapping format. It includes parts of the original reads, that were mapped to a reference genome, together with the position where they belong to. There is an effective binary encoded counterpart called **BAM**.

- <http://samtools.github.io/hts-specs/SAMv1.pdf>

5.3 BED and GFF - Annotations

Annotations are regions in given reference genome with some optional additional information. BED is very simple and thus easy to work with for small tasks, GFF (General Feature Format) is a comprehensive format allowing feature nesting, arbitrary data fields for each feature and so on.

- <http://genome.ucsc.edu/FAQ/FAQformat.html#format1>
- <http://www.ensembl.org/info/website/upload/gff.html>

5.4 VCF - Variants in individuals

VCF stands for Variant Call Format. Given a reference and a set of sequenced individuals, VCF is a format to store the differences in these individuals, compared to the reference, efficiently. There is also a binary counterpart **BCF**.

- <http://samtools.github.io/hts-specs/VCFv4.2.pdf>

Supplemental information:

Course materials preparation

This section contains the steps that we did to produce the materials that course participants got ready-made. That is the **linux machine image**, **online documentation** and the **slide deck**.

6.1 Online documentation

Login to <https://github.com>. Create a new project called *ngs-course-nhrady*, with a default readme file.

Clone the project to local machine and initialize *sphinx* docs. Choose SSH clone link in GitHub.

```
git clone git@github.com:libor-m/ngs-course-nhrady.git
```

```
cd ngs-course-nhrady
```

```
# use default answers to all the questions
# enter project name and version 1.0
sphinx-quickstart
```

Now track all files created by *sphinx-quickstart* in current directory with *git* and publish to GitHub.

```
git add .
git commit -m 'empty sphinx project'

# ignore _build directory in git
echo _build >> .gitignore
git add .gitignore
git commit -m 'ignore _build directory'

# publish the first docs
# setting up argument less git pull with '-u'
git push -u origin master
```

To get live view of the documents, login to <https://readthedocs.org>. Your *GitHub* account can be paired with *Read the Docs* account in *Edit Profile/Social Accounts*, then you can simply ‘import’ new projects from your GitHub with one click. Import the new project and wait for it to build. After the build the docs can be found at <http://ngs-course-nhrady.readthedocs.org> (or click the View button).

Now write the docs, commit and push. Rinse and repeat. Try to keep the commits small, just one change a time.

```
git add _whatever_new_files_
git commit -m '_your meaningful description of what you did here_'
git push
```


This is what it takes to create a basic usable system in VirtualBox. We can shut it down now with `sudo shutdown -h now` and take a snapshot of the machine. If any installation goes haywire from now on, it's easy to revert to this basic system.

6.2.3 Install additional software

R is best used in RStudio - server version can be used in web browser.

```
mkdir sw
cd sw

# install latest R
# http://cran.r-project.org/bin/linux/debian/README.html
sudo bash -c "echo 'deb http://mirrors.nic.cz/R/bin/linux/debian wheezy-cran3/' >> /etc/apt/sources.list"
sudo apt-key adv --keyserver keys.gnupg.net --recv-key 381BA480
sudo apt-get update
sudo apt-get install r-base
sudo R
> update.packages(.libPaths(), checkBuilt=TRUE, ask=F)
> install.packages(c("ggplot2", "dplyr", "reshape2", "GGally", "stringr", "vegan", "svd", "tsne"))

# RStudio with prerequisites
wget http://ftp.us.debian.org/debian/pool/main/o/openssl/libssl10.9.8_0.9.8o-4squeeze14_i386.deb
sudo dpkg -i libssl10.9.8_0.9.8o-4squeeze14_i386.deb
sudo apt-get install gdebi-core
wget http://download2.rstudio.org/rstudio-server-0.98.1081-i386.deb
sudo gdebi rstudio-server-0.98.1081-i386.deb
```

There are packages that are not in the standard repos, or the versions in the repos is very obsolete. It's worth it to install such packages by hand, when there is not much dependencies.

```
# pipe viewer
wget -O - http://www.ivarch.com/programs/sources/pv-1.5.7.tar.bz2 | tar xvj
cd pv-1.5.7/
./configure
make
sudo make install
cd ..

# parallel
wget -O - http://ftp.gnu.org/gnu/parallel/parallel-latest.tar.bz2 | tar xvj
cd parallel-20141022/
./configure
make
sudo make install

# tabtk
git clone https://github.com/lh3/tabtk.git
cd tabtk/
# no configure in the directory
make
# no installation procedure defined in makefile
# just copy the executable to a suitable location
sudo cp tabtk /usr/local/bin
```

6.2.4 Sample datasets

Use data from my nightingale project, subset the data for two selected chromosomes.

```
# see read counts for chromosomes
samtools view 41-map-smalt/alldup.bam | mawk '{cnt[$3]++;} END{for(c in cnt) print c, cnt[c];}' | sort
# extract readnames that mapped to chromosome 1 or chromosome Z
mkdir -p kurz/00-reads
samtools view 41-map-smalt/alldup.bam | mawk '($3 == "chr1" || $3 == "chrZ"){print $1;}' | sort > kurz/00-reads/chr1.txt
parallel "fgrep -A 3 -f kurz/readnames {} | grep -v '^--$' > kurz/00-reads/{}" ::: 10-mid-split/*.fasta

# reduce the genome as well
# http://edwards.sdsu.edu/labsite/index.php/robert/381-perl-one-liner-to-extract-sequences-by-their-
perl -ne 'if(/^>(\S+)/){$c=grep{/^$1$/}qw(chr1 chrZ)}print if $c' 51-liftover-all/lp2.fasta > kurz/20-reads/chr1.txt

# subset the vcf file with grep
# [the command got lost;]
```

Prepare the /data folder.

```
sudo mkdir /data
sudo chmod user:user /data
```

Transfer the files to the VirtualBox image, /data directory using WinSCP.

Now click in VirtualBox main window File > Export appliance. Upload the file to a file sharing service, and use the *goo.gl* url shortener to track the downloads.

6.3 Slide deck

The slide deck was created using Adobe InDesign.

Best practice

This is a collection of tips, that may help to overcome the initial barrier of working with a ‘foreign’ system. There is a lot of ways to achieve the solution, those presented here are not the only correct ones, but some that proved beneficial to the authors.

7.1 Easiest ways to get UNIX

An easy way of getting UNIX environment in Windows is to install a basic Linux into a virtual machine. It’s much more convenient than the dual boot configurations, and the risk of completely breaking your computer is lower. You can be using UNIX while having all your familiar stuff at hand. The only downside is that you have to transfer all the data as if the image was a remote machine. Unless you’re able to set up windows file sharing on the Linux machine. This is the way the author prefers (you can ask;).

It’s much more convenient to use a normal terminal like PuTTY to connect to the machine rather than typing the commands into the virtual screen of VirtualBox (It’s usually lacking clipboard support, you cannot change the size of the window, etc.)

Mac OS X and Linux are UNIX based, you just have to know how to start your terminal program.

7.2 Essentials

Always use `screen` for any serious work. Failing to use `screen` will cause your jobs being interrupted when the network link fails (given you’re working remotely), and it will make you keep your home computer running even if your calculation is running on a remote server.

Track system resources usage with `htop`. System that is running low on memory won’t perform fast. System with many cores where only one core (‘CPU’) is used should be utilized more - or can finish your tasks much faster, if used correctly.

7.3 Data organization

Make a new directory for each project. Put all your data into subdirectories. Use symbolic links to reference huge data that are reused by more projects in your current project directory. Prefix your directory names with 2 digit numbers, if your projects have more than few subdirectories. Increase the number as the data inside is more and more ‘processed’. Keep the code in the top directory. It is easy to distinguish data references just by having `[0–9]{2}` – prefix.

Example of genomic pipeline data directory follows:

```
00-raw --> /data/slavici/all-reads
01-fastqc
02-mm-cleaning
03-sff
10-mid-split
11-fastqc
12-cutadapt
13-fastqc
22-newbler
30-tg-gmap
31-tg-sim4db
32-liftover
33-scaffold
40-map-smalt
50-variants
51-variants
60-gff-primers
```

Take care to note all the code used to produce all the intermediate data files. This has two benefits: 1) your results will be really **reproducible** 2) it will **save you much work** when doing the same again, or trying different settings

If you feel geeky, use `git` to track your code files. It will save you from having 20 versions of one script - and you being completely lost a year later, when trying to figure out which one was the one that was actually working.

7.4 Building command lines

Build the pipelines command by command, keeping `| less -S` (or `| head` if you don't expect lines of the output to be longer than your terminal width) at the end. Every time you check if the output is what you expect, and only after that add the next command. If there is a `sort` in your pipeline, you have to put `head` in front of the `sort`, because otherwise `sort` has to process all the data before it gives out any output.

I prefer 'input first' syntax (`<file command | comm2 | comm3 >out`) which improves legibility, fits better the notion of the real world (plumbing) pipeline (input tap -> garden hose -> garden sprinkler), and when changing the inputs in reusal, they're easier to find.

Wrap your long pipelines on `|` - copy and paste to bash still works, because bash knows there has to be something after `|` at the end of the line. Only the last line has to be escaped with `\`, otherwise all your output would go to the screen instead of a file.

```
<infile sort -k3,3 |
  uniq -c -s64 |
  sort -k1rn,1 \
>out
```

You can get a nice progress bar if you use `pv` (pipe viewer) instead of `cat` at the beginning of the pipeline. But again, if there is a `sort` in your pipeline, it has to consume all the data before it starts to work.

Use variables instead of hardcoded file names / arguments, especially when the name is used more times in the process, or the argument is supposed to be tuned:

```
FILE=/data/00-reads/GS60IET02.RL1.fastq
THRESHOLD=300

# count sequences in file
<$FILE awk '(NR % 4 == 2)' | wc -l
# 42308
```



```
# count sequences longer than  
<$FILE awk '(NR % 4 == 2 && length($0) > $THRESHOLD)' | wc -l  
# 14190
```

7.5 Parallelization

Many tasks, especially in Big Data and NGS, are ‘data parallel’ - that means you can split the data in pieces, compute the results on each piece separately and then combine the results to get the complete result. This makes very easy to exploit the full power of modern multicore machines, speeding up your processing e.g. 10 times. GNU `parallel` is a nice tool that helps to parallelize bash pipelines, check the manual.

Slide decks

Theoretical part (Libor)

Practical part (Vasek)

Other speakers' materials

9.1 Petri Kempainen - LDna

To install the LDna package in the virtual machine you got here, start the machine, log in with PuTTY and run following commands (takes ~10 minutes to build):

```
sudo su    # password is 'user'
apt-get install libcurl3-dev
R --no-save -q <<< 'install.packages(c("devtools"), repos="http://cran.rstudio.com/")'
R --no-save -q <<< '$options(repos=structure(c(CRAN="http://cran.rstudio.com/")))\ndevtools::install_
```

If internet connection does not work in your virtual machine, ask for a new image (Libor or Petri).

If you're interested in installing *LDna* into your very own R/RStudio, you can check the instructions here: <https://github.com/petrikempainen/LDna>.

9.2 Jean Francois Martin

NGS technologies approaches, applications and challenges

Amplicon sequencing by NGS methods