3rd RNA-Seq Workshop Aachen 2015

Andrea Bräutigam and Marie Bolger

Based on the workshop script (RNA-seq Workshop Düsseldorf) by Andrea Bräutigam, Björn Usadel Alisandra Denton and Simon Schliesky

Sponsored by:

Prof. Björn Usadel, RWTH Aachen University, IBMG: Institute for Biology I

and

Version 5 as of March 23h, 2015

Basics

In this section we will start with some basic Unix commands. We will then give an overview on how to run both Perl and Unix programs/scripts. We will briefly introduce the CLC genomics workbench, a commercial graphical user interface which can be used for RNA-Seq analysis.

Unix/Linux: Command-line basics

Unix is a family of computer operating systems which derive from the original Unix developed by AT&T in the early 1970s. Operating systems such as Linux which behave like Unix systems, but do not conform to the Unix specification are commonly known as Unix-like. These include a wide variety of Linux distributions such as Debian, Ubuntu, Fedora and many more. For this course we are using Fedora (Version 19). Most command we will use today are compatible on both Unix and Unix-like systems. We will therefore user the term 'Unix commands' to refer the commands we execute, thought this may not be strictly correct.

Linux provides a variety of Desktop environments (graphical user interface - GUI) which offers the user a 'look and feel' similar to other popular operation systems. These 'Desktop Environments' (we use XFCE for this course) give graphical access to many useful programs and settings via menus bars and icons. There are however many programs, especially in Bioinformatics, which do not provide graphical interfaces and must be executed using text commands.

To begin the course, we will introduce the Unix terminal/shell, which is a text-based interface where the commands are typed and the results of the command are shown. We will also teach you how to use some common commands, and then use these commands to navigate and manipulate files and folders on your machine. Before starting, you should first familiarize yourself with the Unix concepts in the table below, as we will be using these terms frequently

Prompt	This is used to indicate that the shell is ready for you to type in a command. This can vary depending on how your system is configured and which directory you are in – but it typically ends with the dollar sign `$`.
Directory	This is similar to the 'Folder' concept in other operating systems. Directories can hold files or other directories.
Root directory	This is the top level directory and is represented by `/` (a slash). All other directories and files are contained directory or indirectly under the root directory. This structure can be visualized as a tree, with the

	root as the base, directories as branches and files as leaves (although this is typically drawn with the root at the top).
Current working directory	This refers to the directory you are currently in. When you execute a Unix command, by default they work on the contents of this directory. The current working directory is represented by `.` (a dot).
Parent directory	The directory which contains a given directory. This can be visualized as one level nearer the root. The parent directory is represented as `..` (two dots).
Home directory	This is directory for all your stuff. When you first open the terminal shell, this **should** be the 'current working directory'. The home directory is also represented by the tilde `~`. So if you call `cd ~` this will bring you to the home directory.
Path	A path or pathname is a way of referring to a file (or directory) which is not in the current working directory. It consists of the list of directories which need to be navigated to access a file (or directory). Each component of a path is separated with `/` (a slash). A path can be either 'relative' or 'absolute'. A relative path starts from the current working directory, and proceeds to upwards (via parent directories) and/or downwards, to indicate the target file (or directory). Absolute paths begin from the root folder, and are specified by starting the path with `/` (a slash).
Command	A piece of text that tells the computer to do something.
Command argument	Most commands require additional information, such as which input file to use, or where to send the output. These are typed after the command, separated from the command (and each other) by spaces
Command flag	A command flag is typically an optional argument which changes the behavior of a command.

Start the ‘Terminal’ by finding the black box icon on the home screen:

This will open the command terminal and provide you with the command prompt from where you enter the commands.

List the contents of directory you are in (current working directory):

```
1  $ ls
```

```
Desktop     Documents   Download    Examples
```

Here you can see the contents of your home directory, which is where the terminal opens by default.

Change directory to move to the `Desktop` directory:

```
2  $ cd Desktop
```

List the contents of the Desktop directory:

```
3  $ ls
```

Change to the parent directory (the directory above the directory you are in):

```
4  $ cd ..
```

Move to the Examples directory:

```
5  $ cd Examples
```

The `man` command is one of the most useful commands provided by Unix. This gives access to the manual pages of any Unix command you wish to know about. Here you can find out what the command does, what flags it accepts and what argument should/can be provided. If you wish to find out more about the `ls` command, type `man ls` at the command prompt. To exit the manual pages, type the letter `q`. Be aware that manual pages are written in a very terse style, and can be difficult for a beginner to interpret.

Most file and folder manipulations can be carried out using either commands (on the Terminal) or the Linux file manager GUI. If you feel more comfortable performing such operations on the GUI, you can do so.

Find the file manager logo and click on it:

Using the file manager GUI, create two new folders in your home directory called 'results' and 'stuff'. Check on the 'Terminal' to see what has changed.

Change directory to your home folder (the `cd` command without any arguments changes to your home directory):

```
6  $ cd
```

List the contents of this directory:

```
7  $ ls
```

You can now see the two folders you have just created in your home folder – The terminal and GUI are really not that different!

Unix: user permissions

Linux generally gives verbose (thought often not very clear) responses when it cannot run (execute) a program. There may be many underlying reasons for this, but a frequent cause is that the user does not have the permissions to run that program.

How to run/execute scripts in Unix

Quite simply, you just put `./` in front of the 'file' you want to execute. I say 'file' as you can pretty much execute any file (assuming it has the required permissions, of course). Executing a text file will at best output a list of 'command not found' errors – or at worse, find commands in your text which it will happily execute. We therefore typically do not give text files execute permissions. You may wonder why Unix command such as `ls` are not required to have `./` in front of the file(command) (`ls` is a binary file which you can find in `/usr/bin/ls`)? The reason is that the directory `/usr/bin/` is contained in a configuration variable called the PATH. When files are in the PATH, you can execute them with just the name. If you want to see what is in your PATH, just type `echo $PATH`

Generally, regular user accounts do not have system administration permissions (nor should they). If you have administration permissions (or access to the super-user account, often known as the root account), then you can changes permissions for all programs yourself. If you do not have administration permissions to your computer, you need to make sure your system administrator installs and tests that you have permission to execute all programs that you might want to use.

The `sudo` (short for '`s`uper `u`ser `do`') command in Unix allows a normal user to run a command as admin, though they will be required to enter their password before the command is executed. Before a user can use `sudo`, they must first be present in the 'sudoers' list. This needs to be be performed by someone with system administration permissions. Once they are configured to use `sudo`, they need only append `sudo` before the command.

There are several cases where you might want to access files or folders which do not belong to you (and you do not have permissions to do so). The easiest solution if you have `sudo` rights is to run whatever isn't working with the `sudo` command preceding. This gives you temporary superuser-permissions (The highest level of permission you can get on a UNIX system).

> Using the `sudo` command is like telling a 3-year old child to do something. It **WILL** do it no matter how lethal the results will be. Running a malicious script with `sudo` can destroy your computer hardware. Just be sure you know what you are doing. During this course you do not have `sudo` permissions.

Try listing the contents of the root user's home directory:

```
8  $ ls /root
   ls: cannot open directory /root: Permission denied
```

IF you had `sudo` permissions (you currently do not), you could execute the command as follows:

```
9  $ sudo ls /root
   [sudo] password for user:
```

Once the password is entered, the contents of `/root` would be listed.

File/Directory permission in Unix

To see the permissions of files/directory, you need to run the `ls` command with the `-l` flag (you can find out more about the `ls` flags by looking at the man page):

```
10 $> ls -l
```

```
-rw-rw-r--.  3 user1 group1  1605 Feb  7 19:18 example_file1.txt
-rw-rw-r--.  1 user1 group1  1605 Feb  7 19:28 example_file2.txt
drw-rw-r--.  1 user1 group1  1605 Feb  7 23:07 example_directory1
```

The addition of the `-l` flag to the `ls` command provides us with additional information about each file and directory. This information is explained in the table below.

1	-rw-rw-r--	File/Directory permissions
2	3	Number of links (this is beyond the scope of this course)
3	user1	Owner name
4	group1	Owner group
5	1605	File size
6	Feb 7 23:07	Time of last modification
7	example_file1.txt	File/Directory name

The first character is `-` or `d`, `d` indicates that this entry refers to a directory.

The remaining nine characters are in groups of three and refer to the read(`r`), write/modify(`w`) and execute/view contents (`x`) permissions for:

- `o`wner = `rwx` - The owner of the file is indicated by table entry 3
- `g`roup = `rw-` - The group of the file/directory is indicated by table entry 4
- `o`ther = `r--` - This refers to everyone outside of the owner and group (aka the world)

This table indicates permissions for different scenarios.

-rwxrwxrwx	File with full permissions. Everyone can `r`ead/`w`rite/e`x`ecute this file.
drwxrwxrwx	Directory with full permissions. Everyone can `r`ead/`w`rite/`v`iew the contents.
drwx------	Directory with `r`ead/`w`rite/view(e`x`ecute) permissions ONLY for the '`o`wner'. This is typical for your home directory where you do not want others to view/modify your files.

-rwxr-x---	File where the 'user' can read/write/execute. The 'group' members can only read and execute and 'other' has no permission.

To change file/directory permission you need to use the command `chmod`. This command can use a symbolic representation of changes or an octal number representing the number pattern for the new permissions. We will briefly explain both systems.

To change using the symbolic representation, you need to provide:

- whose permissions to change (user, group or other)
- either + to grant the permission or - to revoke the permission
- The permission to grant (read, write or execute)

To change using the octal system, three numbers need to be provided which represent the cumulative permissions for the user, the group and other. The numbers are a binary representation of the `rwx` string.

- r=4
- w=2
- x=1

Simply total the values for the required permission for the user, group and others, to produce the octal permission code (the number 7 signifies full permissions)

Create a New Document in the text editor containing the following two lines:

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

Save the Document as `file_permissions.sh` into your Home Directory.

Move to your home directory and try to execute the script you just created:

```
11 $ cd
12 $ ./file_permissions.sh
```

```
bash: ./file_permissions.sh: Permission denied
```

That does not work, check the file permissions:

```
13 $ ls –l file_permissions.sh
```

```
-rw-rw-r--. 1 user1 group1 0 Mar 11 17:18 file_permissions.sh
```

We can see that this file has read and write permissions for user and group and only read permissions for other. Nobody has permission to execute this file.

Add execute permission for the user:

```
14 $ chmod u+x file_permissions.sh
```

Alternatively you could have changed the permissions using the numbering system

```
15 $ chmod 764 file_permissions.sh
```

Now check again the permissions on your file:

```
16 $ ls -l file_permissions.sh
```

```
-rwxrw-r--. 1 user1 group1 0 Mar 11 17:18 file_permissions.sh
```

We can now see the file has execute permission by user. Now run the script again:

```
17 ./file_permissions.sh
```

This time the script should output the current date and time.

For the following, IT people would bang their heads against a wall or jump off a bridge, but if you work on your own computer, where no one else has access, you might also give all permissions to everyone (even permission to delete the file)

Check the current permissions:

```
18 ls –l file_permissions.sh
```

Grant all permissions to everyone and check again:

```
19 chmod 777 file_permissions.sh
20 ls –l file_permissions.sh
```

Unix: Advanced command-line tools

During our work with large datasets we have come across a few helpful commands that can be used to easily extract information. We will show you a few.

Change directory to the `Examples` directory. Here you should find the files `ReadFile1.fa` and `ReadFile2.fa`

```
21 $ cd Examples
```

To view the `Readfile1.fa` file, we can use the `less` command:

```
22 $ less ReadFile1.fa
```

Press the enter key to advance the file forward one line

Press the space key to advance the file forward one page

Enter `q` to close the file

Sometimes you only want to look at the first or last few lines. To do so, you can use the `head` or `tail` command as follows:

```
23 $ head ReadFile1.fa
24 $ tail ReadFile1.fa
```

The `-n` flag is frequently used with the `head` and `tail` command. This specifies the number of lines to view. To find out more use the `man` command.

To output the complete file in the shell:

```
25 $ cat ReadFile1.fa
```

If you want to look at a file using an application on Windows operation system, do NOT use the default text editor. We recommend Notepad++ (http://notepad-plus-plus.org/). This will open even huge files without freezing or crashing. Notepad++ is also powerful enough to process translations of strings, something that can be really useful to reformat data. We will revisit Notepad++ when we look at extracting biological data.

Before we introduce further commands, you will first need to understand the usage of the `|` (pipe) character. The pipe redirects output from one command into another command. You can think of it as analogous to a conveyor belt in a factory with each machine (command) doing something to the item (data) and passing it on the conveyor belt (via the pipe) to the next machine (command).

Do this `|` with the result do that `|` and with the result of this do the next thing. We can write the results into a file in the same folder by ending the chain with `>` write to this file.txt. If you make Linux do something stupid, such as get stuck in a loop, you can always get out by typing `Ctrl+C`.

To count the number of sequences in a fasta file, we first find every line that has a `>` character using the `grep` (**g**lobally search a **r**egular **e**xpression and **p**rint – a very powerful search utility) command. The output is then passed via the pipe to the `wc` command which counts the lines:

```
26 $ grep “>” ReadFile1.fa | wc -l
```

In this instance, we use the `wc` command with the `-l` flag which will count the lines (appropriate for this purpose, since each sequence name has its own line within the FASTA file). We could also substitute the `-l` flag for `-w` (count the number of words) or `-m` (count the number of characters), which are useful for other purposes

We can use the `wc -m` command to estimate the number of bases in a fasta file. Since we don't want to include the names, we first find every line that does not have a `>` character using `grep -v |` count the number of characters. The `-v` flag used on the `grep` command in**v**erts the search to find non-matching lines.

```
27 grep -v “>” ReadFile1.fa | wc -m
```

If you need to join two or more files together (files that are so large, that copy/paste operations no longer work since they require too much memory), one solution is to output both files, then redirect them to a single file using the 'redirect to file' symbol:

```
28 cat Readfile1.fa ReadFile2.fa > Readfile1_2.fa
```

Note the absence of spaces in the names! Working with spaces is a pain when using the command line (since spaces are also used to separate command arguments). It is best to avoid spaces and other esoteric characters when naming files and folders.

Sometimes we want to change certain regular expressions (strings of characters that repeat) into something else. Text files generated on Macs contain the character `^M` instead of a newline.

First, take a look at the `exampleM.txt` file to verify that these characters are present:

```
29 less exampleM.txt
```

To remove these characters we use the `tr` command. This command translates a given character into another (or if empty, then none) character. We want to translate the `^M` characters to new lines `\n`. You can enter the `^M` character by pressing `Ctrl+V` and then `Ctrl+M`.

```
30 cat exampleM.txt | tr '^M' $'\n' > example_noM.txt
```

Now, take a look at the file again:

```
31 less example_noM.txt
```

The `tr` command is very powerful and can be used creatively for many different replacements (for more information, take a look at the `man` pages).

Blat and tabular Blast output file (or any delimited text) can be parsed using Unix commands.

Look at a Blat Output File from the GUI, note the pattern

Look at an example file:

```
32 less BlatOutput.psl
```

We will now count how many unique 'Reference Identifiers' are in the file. We first remove the column which contains the identifiers (`cut –f14`), sort this column asciibetically (`sort`), omit repeating identifiers (`uniq`), and finally count the number of lines (`wc -l`).

```
33 cut –f14 BlatOutput.psl | sort | uniq | wc –l
```

We will now count many unique 'Reads' are in the file. To achieve this, we use a similar command but this time, we need to `cut` the 'Read' column (`cut -f10`)

```
34 cut –f10 BlatOutput.psl | sort | uniq | wc –l
```

The `uniq` command will not work correctly if the column is not sorted (using the `sort` command).

Many kinds of tabular outputs can be parsed quickly and efficiently using similar commands.

Regular expressions

Regular expressions are an easy way to change repeated patterns in large files using a text editor such as Notepad++ (Windows System) or Textwrangler (Mac OS). With regular expressions you can match a string (a sequence of letters, numbers and signs) and delete it, replace it or change it. Since read files, contig assemblies or read mappings have regular patterns, regular expressions are very helpful especially if you have no knowledge about programming languages.

Learn how to match regular expressions at http://regexone.com/ and test your skills using one of the example files. Alter all read names at once.

Perl-Basics

Perl (probably already overtaken by Python) is a language in which many bioinformatics applications are written. Most biologists cannot read or write Perl (or Python) but fortunately this is not necessary to run Perl scripts. Perl scripts generally have the extension `.pl` and are run on the command line by appending the command `perl`. You can find Perl scripts to do most tasks somewhere on the internet.

Perl is already installed on your machines and you can verify this by running the following command:

```
35 $ perl -v
```

This should return the version, patch level and license information for the installed version.

Now that we have confirmed that Perl is installed, we will look for a script that produces a length histogram of sequences from a fasta file. We will search for this online using the keywords "histogram" "fasta" and "perl". The file that I found as a best hit was called `count_fasta.pl`. If you have found this file (or similar), then download it (don't forget to give/maintain a sensible name including the extention `.pl`) to the folder where your fasta file is waiting for analysis.

```
36 $ cd Examples
```

Once you are satisfied that your script is ready you try to run is as follows:

```
37 $ perl count_fasta.pl ExampleFasta.fa
```

Read the error message that was given, if any. Now, try again to invoke the script, this time with the required parameters:

```
38 $ perl count_fasta.pl –i=100 ExampleFasta.fa
```

The script writes to standard output (your terminal). Try to run it again, this time redirecting the output to a file:

```
39 $ perl count_fasta.pl -i=100 ExampleFasta.fa > FastaHistogram.txt
```

You can look at the documentation for a Perl program on the terminal/shell by calling the `perldoc command`:

```
40 $ perldoc programname.pl
```

Of course this is dependent on the person who wrote the program documenting the code (sadly, this will often not be present, especially if the script was originally written under time pressure).

Here are some starting tips on understanding scripts (you can open the script you just ran as a reference):

- It is possible to run Perl scripts without having to specify `perl` as part of the command. If the first line of the script begins with `#!`, this informs Unix how the file should be run (what command interpreter to use). In this instance it is using `/usr/bin/perl` to execute the script. The `-w` flag instructs Perl to turn on additional warning reporting. Other commonly seen command interpreters are `/usr/bin/python` (for python scripts) or `/bin/bash` (for shell scripts). The script must also be marked with execute permissions for this approach to work. Note that most of the other points below are specific to Perl scripts. As an exercise, copy the `count_fasta.pl` file to `~/bin/` and give it execute permissions, then try to run it without the `perl` prefix (remember that `~/bin/` is in the `PATH`, so will be accessible from anywhere :

```
41 $ chmod 500 count_fasta.pl
42 $ cp count_fasta.pl ~/bin/
43 $ count_fasta.pl -i=100 ExampleFasta.fa > FastaHistogram_test.txt
```

- Everything that starts with a `#` sign (without the additional `!`) are called comments. They are not used in the running of the program, but to help a human reader.
- Most scripts contain a set of comments (typically near the start or end) which indicate who wrote the script and where they come from. It may contain information on what the program does and information on how to run it.
- The statements which start with `use` imports certain modules into the program which the program can then use.
- Everything which starts with `my $something`, `my %something`, `my @something` stores some type of data.
- `foreach`, `while` and `if` start loops which iterate over data.
- `print` outputs something to your screen (or to a file).

Always state the source of programs in your publication. Ideally you should cite the programmer even if you copied the script from an online source.

When you are looking for a script to do a specific task, there are many online resources where you can check. Here are a few tips for using such resources:-

- Forums specific for bioinformatics are a good starting point (seqanswers for instance)
- Always search a forum first for the answer before you post your question. Not doing so suggests that your time is somehow more valuable that the people who reply to these forums.
- Describe your problem/requirement in as much detail as possible. This makes it much easier for people to help you.
- Do not post the same question repeatedly on different message boards (spamming).
- Always be cautious when taking blind advice (or scripts) from forums. Many people posting on these forums are as clueless as you might be. When someone posts ‘well, that worked for me’, it suggest that the person did not understand the problem.
- Become familiar with understanding scripts. Reading them is a much easier task than writing them.
- Be courteous and thank people when they are helpful – it spreads good karma!

CLC-basics

CLC is a GUI driven tool. Originally, CLC was developed for "single gene/protein" applications and the workbench still has those capabilities. It can be used as a substitute for your favorite cloning program.

Since the advent of NGS, CLC has been developing analysis solutions, both for assembly and for gene expression analysis. We have tested CLC and in our hands its performance is comparable to freeware solutions without the installation hassles. But then, it is commercial. Development is ongoing and since we first tested the software, we have noticed major improvements throughout the NGS analyses capabilities. We cannot predict whether this will continue in future.

Disclaimer: We are not affiliated with CLC bio in any way. We are not paid by, nor do we receive discounted software from CLC. The opinions expressed in this workshop reflect our own experiences. We do however receive trial licenses for use at this workshop.

1. Open the CLC genomics workbench by double clicking the CLC symbol on your desktop. The view is instantly familiar.

Experimental Design

Before you run an experiment, consider the following facts. NGS experiments are still very expensive and can rarely be repeated. It is therefore extremely important that you get it right the first time. We have learned the hard way that planning and preparation is time well spent.

To replicate or not – It is tempting to avoid replicating your experiments due to cost considerations. We have performed technical replication and our results were very reproducible. However, biological variation is considerable since RNA-seq is a very sensitive technique. It also needs to be stressed that new technologies don't mean that statistics has become obsolete. The statistical power of RNA-seq experiments performed with a single replicates is severely limited. Would you trust differences in a single leaf measurement on one WT and one mutant plant? Our recommendation is that you divide Illumina lanes by multiplexing or Roche plates by physically separating compartments on the plate. That keeps costs down while giving you biological replicates at the same time. If cost is severely limiting for you, consider (and convince your boss) reducing the number of samples you analyze. Most paper reviewers will reject papers based on the lack of replication.

How many replicates? If your targets are (or may be) regulatory transcripts with expected low read counts, we recommend three or even four replicates. If money is not a problem, and you don't know what to expect, we recommend three. If money is a problem, either perform two replicates or reduce sample number and still do three replicates. There is of course a tradeoff between replication and depth of sequencing. But most of the time Illumina sequencing allows both an adequate depth and replication.

RNA quality – If sequencing fails, the problem is usually in the sample quality. Molecular biology grade RNA is **not** sequencing grade RNA.

Presentation of different sequencing platforms and the RNA and library preparation methods of our lab

.....

.....

.....

.....

Pitfalls to watch out for

- Many mistakes, even those which occurred early in the experiment, will not be detected until you get the sequencing data back. In many cases, the data may be worthless, but the money is already spent. In the best case, it might be possible to recover some

information, but generally this requires massive bioinformatics effort and typically gives less insight than a correctly performed experiment would.

- DNA contaminated RNA samples literally cause nightmares both during assembly and quantification.
- Failed library preps results in loss of replicates. It can make sense to take more samples than you plan to sequence, so you can substitute in a spare sample if an RNA extraction or library preparation fails.
- Overconfidence in a single replicate RNA-Seq experiment causes fruitless time investment since results cannot be confirmed by properly replicated qRT-PCR (i.e. using three biological replicates).
- RNA vanishes during DNAse digests (DNAse contaminated by RNAse).
- RNA contaminations cause sequencing reactions to fail.

Our rule of thumb – Once it is sequenced, it is over. Do not proceed with sequencing unless you are very confident that **everything** was done correctly.

We need to find all mistakes beforehand, so we sometimes spend months optimizing the prep protocols before we go to sequencing. We have also been lucky: The initial sequencing of the Cleomaceae went well and RNA preparation took all of two days – with no quality controls except a bioanalyzer. But we have also been in a project where after spending 100k Euros the RNA turned out to be DNA contaminated – the stuff is still not published although we have already spent months of analysis time. That is not something you like to confess to your boss.

Suggested Workflows for RNA-seq experiments

Comparison within a species, e.g. tissues, treatments, time points.

- Species with a sequenced genome – very short reads possible; assembly not required, may be useful to detect transcript isoforms; read alignment and quantification with fast aligners such as Bowtie possible; essentially like microarrays with more information available and more messy statistics
- Species without a sequenced genome – long to very long reads recommended; assembly required; two read mapping methods available, intra-species on the assembly with a fast aligner such as Bowtie or cross-species on external reference with Blat or Blast; Potentially it is necessary to use long reads for an assembly and short reads to get a more unbiased quantification

Comparison between different species

- Both species without sequenced genomes – long to very long reads recommended; assemblies usually required; two read mapping methods available, both samples cross-species on an equidistant reference such as Arabidopsis or each species onto itself with a fast aligner followed by a determination which contigs corresponds to which contig in the other species;
- One species with a sequenced genome, the other one without – we have not attempted this strategy so far; the difficulty will be that now you look at two factors that influence your read count at the same time, expression level and evolutionary distance with the added problem that low coverage transcripts will yield no to bad contigs and therefore are discriminated against. This approach needs careful thinking through and validity controls.

The following script is guided by approach rather than by workflow. Since each project is different, we have decided to show you what is possible rather than giving you a predetermined recipe.

Assembly

With the ever decreasing cost of sequencing, the amount of data generated from a single sequencing run poses computational challenges. For a *de novo* assembly, the computer has to deal with several million short reads with the objective to create full-length transcripts.

There are two main strategies that assemblies follow:

- Traditional overlap based assemblers
- De Bruijn graph based assemblers.

Presentation: Assembler basics

The major limitation in computational read assembly lies in the memory consumption. Usually, you oversample during an experiment, that is, you have (at least in theory) more reads than you need to get the transcriptome assembled. Read errors and artifactual sequences are additional complicating factors. Therefore the best way to reduce the complexity of the read library is to remove sequences in reads that may contain errors and by removing reads that are too low quality.

Before we start, let's define the starting point. We assume that you start with "de-barcoded, de-primered" sequence files which are what companies usually send you. However be aware that de-primered usually means, there are still some primers left. (Don't trust the data unless you did it yourself!)

Formatting reads: general

Usually Illumina reads come in multiple compressed `*.fastq.gz` files. The `.gz` extension indicates that the files were compressed using the `gzip` command. Compressed `fastq` files are commonly used since they require only a third of the disk space. However, for processing you may need to uncompress them. This step makes the read files more convenient for manipulation.

Open terminal

For libraries smaller than 100M reads:

Go to the directory, where your `.fastq.gz` files are located:

```
44 $ cd ~/Sample001
```

In this directory you should find two `.fastq.gz` files. These files can be uncompressed and concatenated in one step using the `zcat` command:

```
45 $ zcat *.fastq.gz > my_unzipped_reads.fastq
```

Count the number of reads in the new file:

```
46 $ grep "^@HW" my_unzipped_reads.fastq | wc -l
```

Or the following (faster, but output has to be divided by four manually):

```
47 $ wc -l my_unzipped_reads.fastq
```

For large libraries (>100M reads) it is useful to concatenate the reads after the preprocessing steps. This allows the preprocessing of multiple files in parallel (instead of one massive file). This means, of course, that the processing commands need to be run multiple times, once for each file. To make this a little easier, it helps to use a simple file naming convention e.g. `my_reads1.fastq`, `my_reads2.fastq` etc.

Go to the directory where the `.fastq.gz` files are located. Create a new file with the following code, save it, give it execute permissions and execute it. This will uncompress all the files and rename them numerically starting from 1.

```
#!/bin/bash
gunzip *.fastq.gz
#rename the files for structured processing
COUNTER=1
for file in *.fastq; do
mv $file my_unzipped_reads${COUNTER}.fastq
let COUNTER=COUNTER+1
done
```

Attention: if you encounter disk space issues you can unzip 1 file, process it and delete the unzipped file, and then go on with the next file. **Make absolutely sure you have a backup copy (or better several backup copies) of the raw read files somewhere – you may not be able to get them from the sequencing provider again.** The `gunzip` command will uncompress the file in place and not leave the `.fastq.gz` file. Therefore if you delete the `.fastq` afterwards, it will be GONE.

Quality Control

We use the FastQC tool to quickly assess the quality of the raw sequencing data. It is a relatively fast program which offers a user friendly graphical interface and allows the datasets to be quickly evaluated.

To open FastQC, we can called the following command:

```
48 $ fastqc -t 2 --nogroup &
```

> You may have noticed that sometime we use `&` (ampersand) at the end of the command line. This serves to run the command in the background, and therefore you can continue to use the terminal shell and do not have to wait until the program is finished. In this instance, this is sensible given that we are starting a graphical application which you will work with for some time.

After we started the program, we use the graphical user interface to load our dataset. FastQC supports many different fastq formats including `gzip` compressed `fastq` files. After the data has been loaded and analyzed by FastQC, we can go through the different tabs and visualize the data quality. FastQC provides an evaluation for each module and indicates whether the results seem normal (green tick), slightly abnormal (orange triangle) or very unusual (red cross).

Note the FastQC tests are designed for typical DNA sequencing experiments. Other forms of sequence data have characteristics that nearly always trigger the 'red cross' in specific tests. For example, RNA-Seq data will typically have a high level of duplicated reads from the most highly expressed genes, which will be flagged as 'very unusual' although it is normal for this kind of experiment.

Formatting reads: adapter trimming and quality filtering

For trimming the adapters and PCR primers from the reads we will use Trimmomatic. This tool has several different functions which include

- adapter clipping
- quality clipping
- discarding reads below a given quality or length.

Trimmomatic is a modular tool and the user can perform only the functions they require. We perform adapter clipping, removal of low quality bases and reads that are too short to be useful (below 36 bases). Before running Trimmomatic, we first need to know which library preparation kit was used and whether the library was run in paired-end (first command) or single-end (second command) mode. Trimmomatic can also run on compressed files, but in this case, we have already uncompressed them, so will run the tool on the uncompressed files.

If we had Paired End reads, we would use a command like this:

```
49 $ java -jar ~/bin/trimmomatic-0.33.jar PE -basein input_1.fq.gz -
   baseout output.fq.gz  ILLUMINACLIP:../bin/adapters/TruSeq3-
   PE.fa:2:40:12 LEADING:3 TRAILING:3 MAXINFO:40:0.4 MINLEN:36
```

As our reads are Single-End, we use Trimmomatic in the SE mode:

```
50 $ java -jar ~/bin/trimmomatic-0.33.jar SE my_unzipped_reads.fastq
   my_filtered_reads.fastq  ILLUMINACLIP:../bin/adapters/TruSeq3-
   SE.fa:2:40:12 LEADING:3 TRAILING:3 MAXINFO:40:0.4 MINLEN:36
```

`java -jar ~/bin/Trimmomatic-0.33.jar`
This start the Trimmomatic tool. If ran without any parameters, Trimmomatic will give a list of options.

`PE` or `SE`
Paired End or Single End mode

`-basein input.1.fq.gz`
If files are used with the following (standard) naming convention, Trimmomatic will look for the other file automatically. Standard naming conventions are:-

filename_R1_	trimmomatic will look for	filename_R2_
filename_f	trimmomatic will look for	filename_r
filename.f	trimmomatic will look for	filename.r
filename_1	trimmomatic will look for	filename_2
filename.1	trimmomatic will look for	filename.2

`-baseout output.fq.gz`
Trimmomatic generates the four output files (in `PE` mode) based on the root file you give.

`output.fq.gz` will give:-

output_1P.fq.gz	forward paired file
output_2P.fq.gz	reverse paired file
output_1U.fq.gz	forward unpaired file
output_2U.fq.gz	reverse unpaired file

NOTE
In `SE` mode, the `-basein` and `-baseout` flags are not used. Instead the input and output file names (one for each) are provided.

`ILLUMINACLIP:~/bin/adapters/TruSeq3-SE.fa:2:40:12`
The `TruSeq3-SE.fa` refers to the adapter library used. Trimmomatic provides files for the most commonly used adapters and this come with the Trimmomatic zip file.

> `2` seed mismatches - up to how many mismatch bases (out of 16) is still okay. So 3/16 mismatches will prevent a sequence being called an adapter
>
> `40` similar to an e-value. Refers to how good the match needs to be in palindrome mode. Each matched base gets a score of 0.6 In palindrome mode, ~66 matches gives a score of 40

`12` As above but in non-palindrome mode. 20 base matches give a score of 12

`LEADING:3` removes bases at beginning of read of quality is below 3

`TRAILING:3` removes bases at end of read if quality is below 3

`MAXINFO:40:0.4` MAXINFO mode tries to preserve as much of the read as possible even if it means keeping some lower quality bases. This has been shown to give better downstream results.

`40` This indicates that trimmomatic should try to keep reads at least 40 bases long even if it results in bringing in some lower quality bases.

`0.4` This refers to the strictness of the trimming. 0=liberal and 1=strict. This is a trade off between quality and length. For some de-novo assembly, we might use 0.9 here to be absolutely certain to get high quality reads.

`MINLEN:36` If a read does not get even this long, then it should be discarded.

`SLIDINGWINDOW:4:15` Sliding window used the more traditional method of checking the average quality over a given number of bases. If the quality falls under a given threshold, then the read is trimmed from that point.

`4` The window size - how many bases to evaluate for the given quality

`15` The average quality allowed for the given window size.

Open the trimmed file `my_filtered_reads.fastq` again in FastQC and compare to the untrimmed version.

Transcriptome Assembly

If you work with a novel species, an assembly will be required to obtain the gene transcripts. Before creating the assembly, it is highly recommended to trim and filter the reads stringently (using Trimmomatic). Most experiments provide more reads than are needed for assembly, although only genes expressed at a sufficiently high level will be assembled into gene transcripts.

Assembly: CLC

The installation part is explained in great detailed in the CLC manual, so we assume it is already done. Please note that you can use CLC to clean the reads during import. We prefer to trim by using Trimmomatic. If you prefer to use just a single tool to do both tasks, then you can choose to clean the reads during the import process.

1. Start the CLC Genomics Workbench
2. Create a Folder for Sample 1 (Right click on CLC_Data->New->Folder)
3. Import the Data with the following Chain of commands:

   From the toolbar choose NGS Import->Fasta [if cleaning reads in CLC, choose the appropriate format and follow the steps indicated by the program]

   Select Workbench, NEXT

   Browse to [Workshop Directory] select [TestDataSet], NEXT

   select Save, NEXT

   select the folder created in 3., FINISH
4. The imported read library can be viewed by double-clicking the file in the navigation area. The steps for de novo assembly are as follows:

   In the menu click Toolbox->High-Throughput Sequencing->De Novo Assembly

   select Workbench, NEXT

   Select read files and click the ➔ button, NEXT

   Change assembly parameters as desired (use default during workshop), NEXT

   Adjust the mapping (if at all) parameters (create a list of unmapped reads during workshop), NEXT

   Check create Report, select Save, NEXT

   select the Folder created in 3, (optional click +Folder icon and name results), FINISH

5. To get an overview of the assembly open the report file with a double click. The number of contigs and the N50 are shown here.

   To export the assembled contigs, click export in the toolbar, select .fasta in the filetype dropdown menu and choose a folder and filename and click save

Assembly: Trinity

The Trinity suite can also be used to create the assembly. This suite is free to download and use (unlike CLC). It must however be noted that creating the assembly using Trinity is a computationally heavy process, and cannot run on the workshop laptops. It requires 30 minutes on a large server even for this relatively small dataset. We have therefore performed the assembly in advance.

If working with paired-end data, the program can be invoked with the following command:

```
51 $ Trinity --seqType fq –max_memory 50G --left
   my_filtered_reads_1.fastq –-right my_filtered_reads_2.fastq --CPU 12
```

The options explained:

(i) `--seqType fq` you are using fastq files

(ii) `–max_memory 50G` you have up to 50Gb of RAM available

(iii) `--left my_filtered_reads_1.fastq` The name and location of your paired end reads from one end.

(iv) `--right my_filtered_reads_2` the name and location of your paired end reads from the other end.

(v) `--CPU 12` The number of CPUs which should be used by the program.

For single-end data (which is what we are working with), the command line would be:

```
52 $ Trinity --seqType fq –max_memory 50G --single
   my_filtered_reads.fastq --CPU 12
```

Trinity will create a folder for its work (by default called `trinity_out_dir`), and populate it with many output files. The final assembly will be called `Trinity.fasta`. This file has been included in the `Sample001` folder.

Annotation

A *de novo* assembly does not give any functional information about the transcripts which you have found. Annotation of the assembled transcripts using data from a closely related species is recommended. In our plant projects we use *Arabidopsis thaliana* as the reference since we consider it the best curated and annotated plant genome available. All assembled transcripts that cannot be annotated with *A. thaliana* references are then run against RefSeq from NCBI.

1. Start a BLAT run to align the transcripts against the reference

Start mapping with BLAT:

```
$ cd Sample001
$ blat ~/database/TAIR10_cds_20101214_updated Trinity_clean.fasta -
   q=dnax -t=dnax contig2TAIR_mapping.psl
```

2. Extract the best hit for each mapping

Cut columns 1, 10 and 14, sort by ‘query id’ and ‘number of matching bases’ and extract best hit:

```
$ cut -f1,10,14 contig2TAIR_mapping.psl | sort -k2,2 -k1,1nr |
   single_best_blat_hit.pl > best_annotation.txt
```

3. Open a spreadsheet program, e.g. MS Excel or LibreOffice Calculate and load your annotation file
    a. MS Excel: Data -> Insert Text from File -> Choose Tab as separator
    b. LibreOffice Calc: Insert -> Sheet From File -> Choose Tab as separator
4. Load the TAIR functional descriptions file in the same way (file is `~/database/TAIR10_functional_descriptions.clean`).
5. Rename the two sheets:
    a. Right-click on the annotation tab -> rename to ‘annot’
    b. Right-click on the descriptions tab -> rename to ‘desc’
6. Assign the functional descriptions to the identifiers using `VLOOKUP` (Deutsch:`SVERWEIS`)
    a. Switch to the ‘annot’ sheet.
    b. The data should be in three columns which are: ‘number of matching bases’, ‘query’ and ‘target’ (the acutual columns do not have these heading).
    c. Click the empty cell `D1`

d. Type `=VLOOKUP(C1,desc.A2:E41672,5,FALSE)` in cell `D1`. In Excel, the command would be `=VLOOKUP(C1,desc!A:E,5,FALSE)`

The VLOOKUP-function is very powerful, but an error can mess up the whole analysis. The function looks for a specific value (`C1`) within a matrix (`desc.A2:E41672`) and returns the value of a specified column (`5`), that is in the same line as the found search value. The four parameters (in Libre office) are defined as: `VLOOKUP(Search criteria, Search array, Return Column, Sort Order)`

a. `Search Criteria` This is the value which has to be searched for in the determined search range and can either be a value or a cell.

b. `Search Range` This marks the range of cells to be scanned for the search value and the return value.

c. `Return Column` This parameter takes a number that determines the number of the column within the Search Range to return a value from.

d. `Sort order` This is an optional parameter which specifies if it looks for an exact match (`FALSE` or `0`) or the closest match in a sorted list (`TRUE` or `1`).

Attention: Always use `FALSE` as last parameter in the `VLOOKUP` function. Since gene identifiers tend to look very similar, you can easily assign the wrong descriptions to your contigs by using `TRUE`.

7. Now cell `D1` should show the first gene description. You can now copy the formula to the rest of the lines by

   a. Copy and pasting (tedious for over 40000 entries).

Double-clicking on the black box in the lower-right corner of the cell (this will also take a long time!!!) This step is optional in the workshop. You can find the already find the file `Trinity_annotations.ods` in the `~/Samples001`. You can therefore open the file and skip to step 10.

Attention: The Program will recalculate the cells over and over, whenever you change anything in any cell. Therefore it is highly recommended to select all cells containing VLOOKUP (click on the column header `D` to select the whole column), copy them, and paste them as values instead of formulas (Right-click -> Paste Special -> Values only).

8. Clean Up: After replacing the formulas by values, you can safely delete the ‘desc’ sheet to reduce the file size.

9. Save the file as `Trinity_annotations` as an ODF Spreadsheet (`.ods`).

10. Delete columns `A` and `C` and save the file as a 'CSV file' (call the file `Trinity_annotations_trimmed`), using 'Tab' as a delimiter.
11. Use the Perl script `AppendHeaderToFasta.pl` to translate the generic fasta headers to a more meaningful header.

First we will check the usage of the script:

```
56 $ perldoc AppendHeaderToFasta.pl
```

Run the script according to instructions and save the output.

```
57 AppendHeaderToFasta.pl Trinity_annotations_trimmed.csv
   Trinity_clean.fasta > annotated_sequences.fasta
```

Take a look at the results.

Assembly parameters and quality

Most NGS papers will start (and some even end there) with descriptive data tables. These need to be generated although they are rather boring from a biologist's point of view. A number of parameters are generated by the `count_fasta.pl` script: number of contigs, N25, N50, N75, GC content and a length distribution.

Some parameters are generated by annotation against a reference:

How many contigs match the reference?

How many unique reference sequences are covered?

How often each reference sequence is covered (gives an idea about the assembly quality)?

A second question is how good the assembly is? It is difficult to give a final answer to this question but the following parameters will help:

Number of contigs:

The number of contigs cannot be applied as a direct measure for the assembly quality. It is however still possible to compare it to the number of expected transcripts. How many transcripts do I expect? Unfortunately there is no golden rule for this question. Make an educated guess! To give an example: A leaf transcriptome is thought to consist of roughly 50 - 60% of the plant's genes. Therefore in Arabidopsis one would expect the leaf transcriptome to have approx. 20,000 different transcripts.

To count the number of transcripts, `grep` the lines preceded by "`>`" from your transcripts

```
58 $ grep “^>” Trinity.fasta | wc -l
```

Number of contigs, N25, N50, N75, GC content, a length distribution:

One program which gives a fairly comprehensive parameter list is the `count_fasta.pl` script which you used in the introduction.

Run the Perl Script `count_fasta.pl` as described earlier on two example files and write the output to a file:

```
59 $ count_fasta.pl –i=100 ExampleFile1.fa > ExampleFile1_histogram.txt
60 $ count_fasta.pl –i=100 ExampleFile2.fa > ExampleFile2_histogram.txt
```

Copy the result into Libre office (or Excel) and visualize the histogram. For comparison, it is useful to also plot a histogram of Arabidopsis transcripts.

If you compare two samples or closely related species, the parameters should be roughly similar.

You can run the same program on a reference transcriptome from a closely related species with a sequenced genome. Under the assumption that the genomes are reasonable similar, you can use the values generated from the reference as benchmarks for your assembly.

How many contigs match any reference?

How many unique reference sequences are covered?

How often each reference sequence is covered?

To identify these parameters mentioned, you need a tabular (!) Blast or BLAT output

Output Format of a Blast Table:

column number	Description	comments
1	Query	sequence you used to search
2	Subject	sequence you in reference database
3	% identity	identity along the match (not between complete sequences)
4	alignment length	
5	number of mismatches	
6	number of gap openings	
7	start of matching query sequence	
8	end of matching query sequence	
9	start of matching subject sequence	
10	end of matching subject sequence	
11	e-value	careful, depends on reference database size!
12	bit score	independent of reference database

Output Format of a Blat Table:

column number	Description	comments
1	Match	Number of matches
2	Mismatch	Number of mismatches
3	Rep. match	Number of matches that are part of repeats
4	N's	Number of N's in alignment
5	Q gap count	Number of gaps in query
6	Q gap bases	
7	T gap count	Number of gaps in target
8	T gap bases	
9	Strand	
10	Q name	Name of query sequence
11	Q size	
12	Q start	
13	Q end	
14	T name	Name of target sequence
15	Tsize	
16	T start	
17	T end	
18	Block count	
19	Blocksizes	
20	qStarts	
21	tStarts	

Extract the number of queries with a match (assembled transcripts/contigs found in reference) in the Blast output file `contig2TAIR_mapping.txt` (we will explain later how to run Blast from the command line):

```
61 $ cut –f1 contig2TAIR_mapping.txt | sort | uniq | wc –l
```

Extract the number of matching references:

```
62 cut –f2 contig2TAIR_mapping.txt | sort | uniq | wc –l
```

Find out how many times each reference sequence was matched:

```
63 cut –f2 ExampleBlastResult.txt | sort | uniq –c >
   ExampleBlastResult_frequencyofmatches.txt
```

Open the table in a spreadsheet program and sort by the number of matches then inspect the table manually.

Using the `countif` (`zählenwenn`) function, test how many references are matched more than 100, more than 50, more than 10, more than 5, more than 1 times.

In our experience, genes that are highly expressed yield many contigs in the assembly. For a typical leaf transcriptome assembly, the worst offenders are LHC and photosynthesis genes. For you experiments, it may be others. Generally, with a smaller dynamic range of your sample, this is less of a problem.

Hybrid transcripts While you want complete contigs for your transcriptome reference database you also want the contigs to contain sequence from only one transcript. However, since Murphy's law applies to assemblies too, there will be contigs that consists of two or more transcripts which we call hybrids. Simon Schliesky has developed a script that can recognize these hybrids (if only in Arabidopsis yet). The FASTA-like output makes counting easy.

This script needs the BLAT contig mapping (`contig2TAIR_mapping.psl`) which was created in the annotation section.

Sort the `contig2TAIR_mapping.psl` file by 'query' and 'number of matching bases' (reversed) and pipe the results to the Perl script:

```
64 $ tail –n +6 contig2TAIR_mapping.psl | sort -k10,10 -k1,1nr |
   detect_hybrids.pl > hybrids_contig2TAIR.txt 2>/dev/null
```

We have appended `2>/dev/null` to the command to suppress the debugging output. This redirects the output to a 'null' device which is used for disposing of unwanted output streams.

You can use the `head` command to check whether the script worked:

```
65 $ head hybrids_contig2TAIR.txt
```

Confirm that this look similar to the output example:

- Each contigID line begins with `>`
- Every contig that mapped only to one target is tagged 'not a hybrid'

Count how many contigIDs were processed:

```
66 $ grep ">" hybrids_contig2TAIR.txt | wc -l
```

Count the IDs that are 'not a hybrid':

```
67 $ grep ">" hybrids_contig2TAIR.txt | grep "not a hybrid" | wc -l
```

Counting the IDs that are hybrids (just reverse the second `grep` with the `-v` flag):

```
68 $ grep ">" hybrids_contig2TAIR.txt | grep "not a hybrid" -v | wc -l
```

Our list of scripts to quality control an assembly can be found in the following paper: http://journal.frontiersin.org/Journal/10.3389/fpls.2012.00220/full

Using subsets of .fasta file

Sometimes it is desirable to extract a specific set of contigs to work on. For example, you might want to extract all contigs which match *A. thaliana*. If you have a list of the contig identifiers, you can extract the sequences from a fasta file. If you want to extract all contigs that map to chromosome 1 of the *Arabidopsis* reference:

1. Open the annotation file created in the annotation part with Libre Office (`Trinity_annotations.ods`)
2. Sort by Target column (C)
3. Select all query identifier (column B) that match `AT1GXXXXX`
4. Right-click->copy
5. Open text editor
6. Paste the identifiers (should be one per line now)
7. Save the file to as `ids_to_extract.txt`
8. Open Terminal
9. Run the `Extract_Sequence_by_ID.pl` script as follows:

```
69 $ extract_sequence_by_id.pl --ID=ids_to_extract.txt  --
   sequences=Trinity_clean.fasta --fileformat=txt --outfile=subset.fasta
```

10. [optional] An additional parameter is the `--exclude` if you rather want the sequences **NOT** mentioned in the file `ID=xxx`
11. [optional] The `fileformat` parameter can also be set to `fasta`, in case your identifier are preceded by a `>`, when grep'd from different fasta file for example.

Gene Expression

There are two fundamentally different gene expression quantification experiments:

(i) When you have sequenced a species and you want to map the reads TO A DIFFERENT SPECIES – use Blast or Blat.

(ii) you have sequenced a species and want to map the reads TO THE SAME SPECIES, either the genome reference or assembled contigs – use BWA, Bowtie (or one of its many applications) or CLC bio.

Presentation: Principles of read mapping algorithms

Read mapping: local BLAST

The most traditional software for mapping reads to a reference is the Basic Local Alignment Search Tool, short BLAST. It essentially takes two `.fasta` files as parameters. One is the read file, the other the database, which the reads will be aligned to. BLASTing your reads with a BLAST web interface would take quite long and would not be very comfortable given the number of sequences you are dealing with. Thus you will learn to use a local BLAST on your computer in this section. (The installation of Blast is described in the appendix)

1. To create a blast database for Arabidopsis, normally you would need to download the sequences from Arabidopsis.org, but for time reasons we have already downloaded it for you. To download the file, you could use the command `wget`. Check out what the command does using the `man` command.

```
70 $ wget
   ftp://ftp.arabidopsis.org/Sequences/blast_datasets/TAIR10_blastsets/T
   AIR10_cds_20101214_updated
```

Instead, just change to the database directory directly in your home directory:

```
71 $ cd ~/database
```

Format as nucleotide database to allow it to be used with BLAST:

```
72 $ formatdb -pF -i TAIR10_cds_20101214_updated
```

`-pF` This means that it is not a protein alphabet `F`=False

2. Prepare the read file (BLAST takes `.fasta` files only). As the target in read mapping is to get quantitative information, you wouldn't want to map your duplicate removed `.fasta` from the assembly part.

Change directory to the `Sample001` directory:

```
73 $ cd ~/Sample001
```

Use the script `fastq_to_fasta` to convert your trimmed reads to the fasta format:

```
74 $ fastq_to_fasta -Q33 -i my_filtered_reads.fastq -o
   my_filtered_reads.fasta
```

You're dealing with small test datasets in this workshop, that's why we concatenate the single .fasta files. However, when dealing with very large datasets, it might be preferable to work on the individual files in parallel and merge the information at the result level (after mapping)

3. Although the list of possible blast parameters is very long, it is sufficient to supply a handful:
    a. `-d database` the reference database to BLAST against
    b. `-i query_file` the query read file
    c. `-e 0.0001` e-value default is `10` (here `0.0001` or 10^-04)
    d. `-m8` output format (`8`: tabular ouput, 1 line per match)
    e. `-a N` use multiple=N CPUs (here: `-a6`)
    f. `-FF` deactivate read masking
    g. `-p algorithm` either `blastn`, `blastp`, `blastx`, `tblastp` or `tblastx` (here: `tblastx`) explanation see: http://www.ncbi.nlm.nih.gov/BLAST/blast_program.shtml
    h. `-o output_file`, the result file
    i. `- b1` only one hit

The following Blast command would take too long on the Workshop laptops. We have therefore already created the output file which would be generated form this command. This can be found in `~/Sample001/mapping001/Sample001_to_TAIR10cds.txt`.

The following command would start BLAST (all written in one line! The multiline version in this script is just for ease of reading):

```
75 $ blastall -d ~/database/TAIR10_cds_20101214_updated
              -i ~/Sample001/my_filtered_reads.fasta
              -e 0.0001
              -m8
              -a6
              -FF
              -p tblastx –b1
              -o mapping001/Sample001_to_TAIR10cds.txt
```

4. Parsing the BLAST output is covered in the "Counting reads" section

Change to the mapping directory:

```
76 $ cd mapping001
```

Use the `head` command to see how the result file looks:

```
77 $ head Sample001_to_TAIR10cds.txt
```

Use `cut` to remove the desired columns ('query', 'target' and 'e-value'):

```
78 $ cut -f1,2,11 Sample001_to_TAIR10cds.txt >
   cut_Sample001_to_TAIR10cds.txt
```

Use sort to `sort` the results by 'query id' and then `n`umerical `r`everse by 'e-value':

```
79 $ sort -k1,1 -k3,3nr cut_Sample001_to_TAIR10cds.txt >
   sort_Sample001_to_TAIR10cds.txt
```

Run the script `single_best_blast_hit_only.pl` to narrow down the mapping to one match per read:

```
80 $ single_best_blast_hit_only.pl sort_Sample001_to_TAIR10cds.txt >
   best_Sample001_to_TAIR10cds.txt
```

Count the number of reads by `cut`ting the 'target' column and counting frequency of unique IDs:

```
81 $ mkdir Count001
82 $ cut -f2 best_Sample001_to_TAIR10cds.txt | sort | uniq -c | awk
   '{print $2"\t"$1}' > readcounts_Sample001_to_TAIR10cds.txt
```

Read mapping: BLAT

BLAT (BLAST Like Alignment Tool) is a tool which operates faster with read-mapping than its older brother BLAST. It is suitable for contig annotation and read mapping of small libraries (less than 1million reads)

1. Running BLAT requires a few parameters
    a. the reference database for alignment
    b. the query file containing the reads
    c. `-q=X`, the alphabet of the query (here: `-q=dnax`)
    d. `-t=X`, the alphabet of the database (here: `-t=dnax`)
    e. `output_file`, the result file ending in `.psl`

The following BLAT command would take too long on the Workshop laptops. We have therefore already created the output file which would be generated form this command. This can be found in `~/Sample001/mapping001/Sample001_to_TAIR10cds.psl`.

```
83 $ blat ~/database/TAIR10_cds_20101214_updated
   ~/Sample001/my_filtered_reads.fasta
        -q=dnax -t=dnax mapping001/Sample001_TAIR10cds.psl
```

2. Parsing of BLAT output is covered in the "Counting reads" section.

Counting reads: BLAST/BLAT

Blast and Blat are much slower than Bowtie but they tolerate many more non-matching bases in the reads vs. the reference. The output returns not only the best hit but all hits within the matched parameters. Therefore we developed two parsing scripts, one for 'BLAST -m8' and one for 'BLAT' output files. Both extract from a sorted list of mappings only the best hit for each query. With Blast the best hit = lowest e-value. In the case of BLAT best hit = highest number of matching bases.

1. Parsing of the BLAT/BLAST output is done with Unix command-line tools and some Perl. First we reduce the amount of data to handle, by extracting the columns we need for further analysis.

BLAST Specific steps:

Extract columns 1, 2 and 11 which correspond to 'query', 'target' and 'e-value' and write these to a new file:

```
84 cd
85 mkdir counts001
86 cut -f1,2,11 ~/Sample001/mapping001/Sample001_to_TAIR10cds.txt >
   ~/counts001/cut_Sample001_BLAST.txt
```

Sort the file first by 'query' (column 1) then by 'e-value' (column 3) and output these to a new file:

```
87 sort -k1,1 -k3,3nr ~/counts001/cut_Sample001_BLAST.txt >
   ~/counts001/sort_Sample001_BLAST.txt
```

BLAT specific steps:

Extract columns 1, 10 and 14 which corresponds to the 'number of matches', 'query' and 'target' and output them to a new file:

```
88 tail -n +6 ~/Sample001/mapping001/Sample001_TAIR10cds.psl | cut -
   f1,10,14 > ~/counts001/cut_Sample001_BLAT.txt
```

Sort the file first by 'query' then by 'number of matching bases' and output this to a new file:

```
89 sort -k2,2 -k1,1nr ~/counts001/cut_Sample001_BLAT.txt >
   ~/counts001/sort_Sample001_BLAT.txt
```

Run the Perl script `single_best_blat_hit.pl`:

```
90 $ cd ~/counts001/
91 single_best_blat_hit.pl sort_Sample001_BLAT.txt >
   best_Sample001_BLAT.txt
```

Steps for both BLAT and BLAST:

From the resulting file you can count the reads by counting the occurrence of unique target IDs. In other words, if one ID was hit by 50 reads, it is 50 times in the list. Therefore the command `uniq -c` will return 50.

Remember to ALWAYS `sort` before calling `uniq –c`:

```
92 cut -f2 sort_Sample001_BLAST.txt | sort | uniq -c >
   ~/counts001/readcount_Sample001_BLAST.txt
```

for BLAT use `cut -f3` and the BLAT mapping file

2. For import to MS Excel or LibreOffice set white-space and tabulator as separator and check "merge separators".

Read mapping: Bowtie2/Tophat2

Bowtie2 is a very fast and powerful short read aligner, which can be used to map directly to the genome and pick up splice variants when used with the Bowtie2/Tophat2/Cufflinks pipeline. As with all fast aligners, Bowtie2 can only tolerate a small number of mismatches.

1. The first step is to build a reference index for Bowtie2.
    a. first argument is a fasta file (generally of the full genome, or contigs)
    b. second argument is the root name of the index to be created

```
93 $ cd ~/database
94 $ bowtie2-build TAIR10.fa TAIR10.index
```

2. The Tophat2 program repeatedly calls Bowtie2 in a robust method to allow mapping across splice junctions. Performance is improved if the genome annotation file (`.gff/.gff3`) is supplied.
    a. `--b2-very-sensitive` is the most sensitive standard setting
    b. `-p` specifies the number of cores
    c. `-o` specifies the output directory
    d. `-G` specifies the gff annotation file for the genome. Bowtie will construct another transcriptome-specific mapping index from this.

e. `--transcriptome-index` tells bowtie to save this transcriptome index under the given name for later usage. (-G does not need to be called for later runs against same genome)

```
95 $ cd ~/Sample001
96 $ tophat2 --b2-very-sensitive –p2 -o Sample001.topout -G ~/database
   /TAIR10_GFF3_genes.gff –-transcriptome-index=../database/TAIR10_trans
   ~/database/TAIR10.index my_filtered_reads.fastq
```

f. the penultimate argument specifies the path to the Bowtie index made above

g. the final argument is the reads file(s)

3. The reads are now mapped, but this pipeline allows us to get closer yet to a user-friendly output using cufflinks. Cufflinks assembles the transcripts, from which the reads were most likely to have been generated and provides the user directly with the FPKM (fragments per kb per million ~ RPKM)

a. `-o` Specifies the output directory

b. `-u` Tells Cufflinks to do an initial estimation procedure to more accurately weigh reads mapping to multiple locations in the genome. See How Cufflinks Works for more details.

c. `-b` Genome fasta file so that cufflinks can perform bias correction

d. `-G` Specifies the gff annotation file for the genome

e. The final argument is the mapped reads (`.bam`) file produced by Tophat

```
97 $ cufflinks -o Sample001.cuffout -u -b ~/database/TAIR10.fa -G
   ~/database/TAIR10_trans.gff Sample001.topout/accepted_hits.bam
```

Take a look at the `genes.fpkm_tracking` file under `Sample001.cuffout`

Bowtie output can be directly visualized with Tablet, a freeware available at http://bioinf.scri.ac.uk/tablet/index.shtml. You can use Bowtie not only for mapping to complete databases but also for looking at a single reference gene or contig by altering the reference in analogy to the process described above. If you use it that way, you can directly look at the output in a GUI using tablet.

Cufflinks has implemented differential expression testing on the raw mappings in the program cuffdiff, which allows for both robust and (comparatively) easy statistics when using this pipeline.

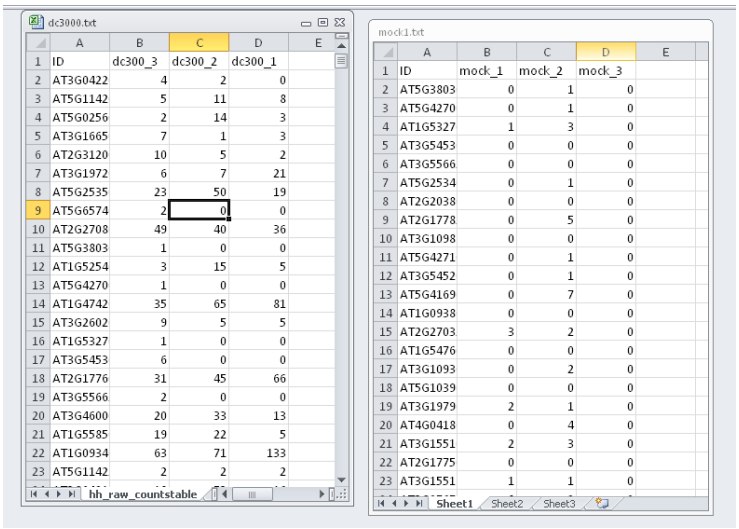
Read mapping and counting with CLC

1. Install the CLC annotation plug-in.
2. Load the reference sequences from http://www.arabidopsis.org/
3. Download the gff file from http://www.arabidopsis.org/
4. Annotate the reference sequence in CLC using the plug-in
5. Using CLCs RNA-seq tool, map the reads to the reference
6. Inspect the result table and choose the desired outputs.
7. Export the result table for further analysis

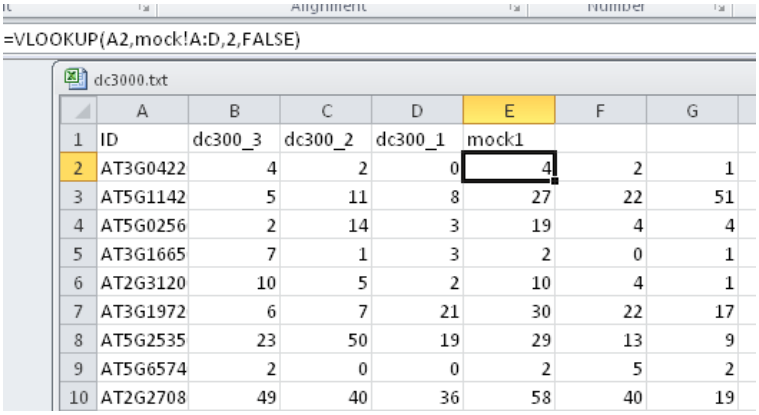
Statistical evaluation of readcounts

In this section we are going to use the R package introduced in the basics section. For the statistical comparison of gene expression data, we need at least two datasets. Therefore we prepared a file called `Dc3000.txt` and `mock.txt`. The readcount files created in the counting-reads section are formatted in an inappropriate way for the R Script: variable number of spaces, number of reads, single space and identifier. We are now going to reformat that file to: identifier, tab, number of reads. (As we need some replication both files already give data for three replicates)

1. Import both files separately into excel



2. Copy mock into the second tab of dc3000 and name the sheet mock
3. select E2 and enter VLOOKUP formula: `=VLOOKUP(A2,mock!A:D,2,FALSE)`


=VLOOKUP(A2,mock!A:D,2,FALSE)

dc3000.txt

	A	B	C	D	E	F	G
1	ID	dc300_3	dc300_2	dc300_1	mock1		
2	AT3G0422	4	2	0	4	2	1
3	AT5G1142	5	11	8	27	22	51
4	AT5G0256	2	14	3	19	4	4
5	AT3G1665	7	1	3	2	0	1
6	AT2G3120	10	5	2	10	4	1
7	AT3G1972	6	7	21	30	22	17
8	AT5G2535	23	50	19	29	13	9
9	AT5G6574	2	0	0	2	5	2
10	AT2G2708	49	40	36	58	40	19

4. select F2 and enter VLOOKUP formula: `=VLOOKUP(A2,mock!A:D,3,FALSE)`

5. select G2 and enter VLOOKUP formula: `=VLOOKUP(A2,mock!A:D,4,FALSE)`

6. drag the whole columns down so you get values for all

7. Save file as tab separated txt format by clicking “Save as”, choosing “tab separated text (`*.txt`)” from the dropdown menu and naming it “`counts.txt`”

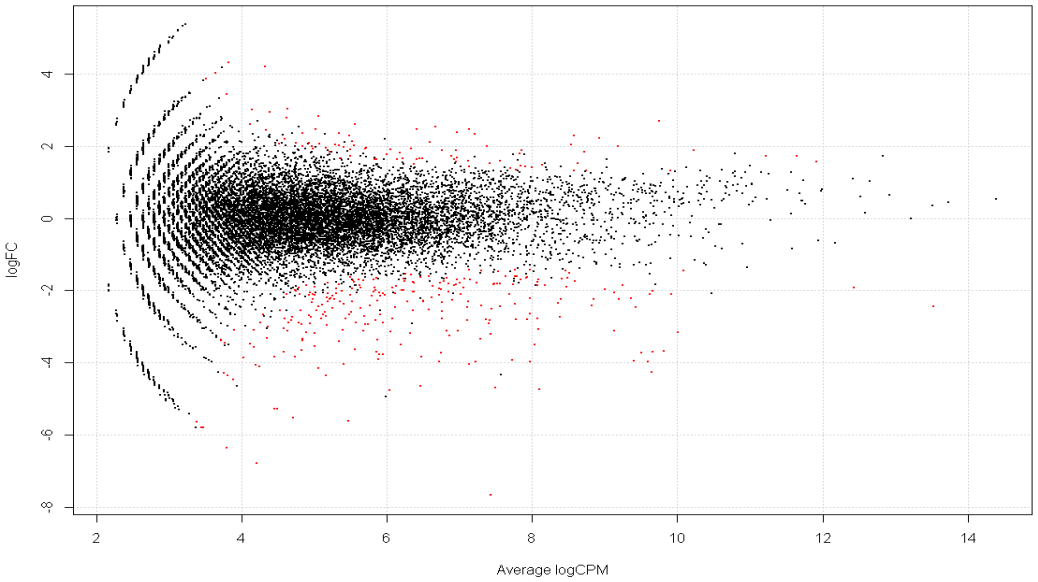
The reformatting shows two major changes, first the order of columns is changed in comparison to the readcount file and second the samples are merged into one file. We could have done this is R but it would not have been so easy.

```
98 > library(edgeR)
99 > x <- read.table("counts.txt",row.names=1,header=T,sep="\t")
100> group <- factor(c("dc","dc","dc","mock","mock","mock"))
101> y <- DGEList(counts=x,group=group)
102> y <- calcNormFactors(y)
103> y <- estimateCommonDisp(y)
104> y <- estimateTagwiseDisp(y)
105> et <- exactTest(y)
106> topTags(et)
107> summary(de <- decideTestsDGE(et, p=0.05, adjust="BH"))
108> detags <- rownames(y)[as.logical(de)]
109> plotSmear(et, de.tags=detags)
110> write.table(file="results.txt",sep="\t",topTags(et,n=length
   (rownames(y))))
```

1. The `library()` command loads modules to extend the functionality of R

2. Remember R uses `<-` to store information into variables

3. `read.table()` reads in a file
4. `group` tells R what kind of biological replicates you used
5. `DGEList` is just a better way of representing your data
6. `EstimateXXXDispersion` tries to find the overdispersion
7. `exactTest` is just conducting an exact test
8. `decideTests` just tells us how many genes are differentially expressed
9. `plotSmear` makes the following plot in which DE genes are highlighted



10. We then write the file to a table which we can open in Excel (be careful the open left cell needs to be shifted by one)

DD.txt

	A	B	C	D	E
1	logFC	logCPM	PValue	FDR	
2	AT4G1250	-4.26986	9.640145	5.16E-20	1.10E-15
3	AT2G1919	-4.69092	7.497199	2.35E-19	2.51E-15
4	AT2G3920	-3.94521	9.397381	6.45E-19	3.59E-15
5	AT5G4843	-7.67811	7.435818	6.72E-19	3.59E-15
6	AT3G4628	-4.73452	8.102571	6.33E-18	2.70E-14
7	AT1G5180	-3.93823	7.725644	4.13E-17	1.47E-13
8	AT5G6412	-3.67076	9.812979	6.27E-17	1.91E-13
9	AT2G4522	-3.96389	7.966681	3.19E-16	8.50E-13
10	AT2G3953	-4.65478	6.46064	4.37E-16	1.04E-12
11	AT2G3938	-4.75608	6.044068	8.00E-15	1.71E-11
12	AT4G1249	-3.71965	9.503719	1.65E-14	3.20E-11
13	AT2G4362	-3.15641	10.00781	2.39E-14	4.25E-11
14	AT3G5515	-5.61547	5.47279	3.20E-14	5.26E-11
15	AT1G3072	-4.03465	7.133514	2.02E-13	3.09E-10
16	AT3G2206	-3.11294	9.131588	4.32E-13	6.14E-10
17	AT5G3958	-3.71984	6.747885	6.12E-13	8.17E-10

Remember: R understands multiline commands, if you write code in R, take as much space as you like. Code that is easy to read - even typographically - is easy to change.

If the R script runs without throwing an error (look at the console), you will find a result directory in the current working directory (set by you with `setwd(…)` command). It will contain all the graphs, you could see in RStudio, exported as either `.jpg` or `.png`. It will also contain a text file giving a detailed output of differential gene expression.

Biological Interpretation

Similar to the early days of microarray experiments, RNA-seq experiments can currently still be published on their own, especially if non-model species are analyzed. A successful manuscript tolerates about 20% speculation (interesting single genes, new hypotheses) if 80% of the manuscript consists of solid data which fits into previous work.

To provide the 80%, a set of testable hypotheses is needed. Similar to classical wet bench experiments, the hypotheses are derived from previous experiments or published work. As an example, the comparison between a green and a non-green tissue RNA-seq experiment is chosen. The tests can be transferred to any dataset comparison.

Example 1: RNA-Seq was performed on Leaf and root samples from maize and then compared: In the green data set, the photosynthetic genes are expressed; in the non-green data set, the photosynthetic genes are not expressed

Test:

i) create a functional annotation of the genes using a reference of choice (Mapman categories, GO terms, KEGG terms),

ii) use sorting and/or Excel functions to count the number of photosynthetic genes expressed in each dataset -> reject of support the initial hypotheses [based on your interpretation of the data]

iii) formalize the test by testing enrichment (using Chi-square or Fishers Exact Test) of photosynthetic genes in the green compared to the non-green dataset [formally reject or support initial hypothesis]

Visualization:

i) reformat the datatable according to the needs of Mapman

ii) visualize the photosynthetic category

Create three additional hypotheses about the dataset which are testable similar to the example above.

.....

.....

.....

.....

Creating an Functional Annotation

The objective is to produce an annotation table from publicly available data sources; the example is Arabidopsis but will work for any species or reference sequence collection. We routinely annotate our species with Arabidopsis. The annotation of the transcriptome databases links to the closest Arabidopsis homologue and we transfer the information gathered as described below.

1. Get an annotated reference database that is hierarchical (that means NO GO terms), example Mapman categorization (http://mapman.gabipd.org) For the Mapman Table we need to reformat the table first: Reorder the columns into IDENTIFIER; DESCRIPTION; NAME. Use the text to columns tool in the spreadsheet program to split the NAME column at each point. Rename the headers into Mapman1, Mapman2,… Voila, you got yourself a basic annotation
2. Reopen the text file in the spreadsheet program and copy it in a new sheet in the original annotation table.
3. Make sure that the Identifiers in both sheets are exact matches (including capital letters). Use VLOOKUP (SVERWEIS in german) for string pattern matching: =VLOOKUP(Lookupvalue; SearchMatrix; column for return value; FALSE) in this example =VLOOKUP(A2;2ndSheet!A:B;2;false). Fill the column. Copy the column and reinsert values only.

Any information that is organized as a table can be imported into the annotation. Examples are transcription factor annotations from a review, results from a localization prediction program, really anything.

You can also use the MapMan online service for automatic annotation called Mercator. http://mapman.gabipd.org/web/guest/app/mercator. Here you can submit a whole transcriptome (AFTER assembly). But this would take too long within the course.

You can also map GO terms into the table. Since these are NOT hierarchical terms, you cannot use them easily for the follow-up analysis suggested below. To use GO terms, we recommend using one of the many JAVA based enrichment test tools such as AgriGO (http://bioinfo.cau.edu.cn/agriGO/index.php) for enrichment tests.

The way you annotate your table will heavily depend on what you want to analyze and on how much is known. While *A. thaliana* is the best annotated reference, it may not be suitable for non-dicot plants. Whatever you use, make sure that it is a species whose genome is complete so that your reference is

complete as well. Making your own annotation for a whole genome (apart from copying in information already provided as we did today) will take weeks to months.

The annotation table can be used for both analyzing the assembly and for analyzing a read frequency table.

Assigning group membership using Excel

Locate the photosynthetic category/categories included in the Mapman annotation. Using Excel functions (`=COUNTIF`, `=COUNTIFS`), count the number of genes annotated as photosynthetic, of genes annotated as photosynthetic and of genes annotated as photosynthetic and expressed in green tissue.

Construct a table:

	photosynthetic	**non-photosynthetic**	**sums**
Found in Green Tissue			Sum of photosynthetic and not photosynthetic
Not found in Green Tissue			All non found in green tissue
Sums	All photosynthetic	All non photosynthetic	Grand Sum

Excel sets its tables for a Chi Square test up in a different way!

Generally these tables always look like this, they behave a little bit like a Soduku, you only need a few numbers to fill out the whole table

	Property A	**Not property A**	**All**
Property B	U	V	U+V
Not Property B	X	Y	X+Y
Sums	All with property A (U+X)	All not having property A (V+Y)	(U+X)+(V+Y) OR (U+V)+ (X+Y)

Calculate the p-value using the chi-square test, using the Fishers Exact Test and check whether the enrichment of photosynthetic genes expressed in green tissue is significant.

You can find an online version of the Fisher test here.

http://www.quantitativeskills.com/sisa/statistics/fisher.htm

The key to successful data evaluation is producing hypotheses before and while looking at the data. To generate the 80% it is important to look at pathways and gene groups and to NOT look at single genes.

An example of a hypotheses-driven RNA-seq manuscript (which is worded in terms of hypotheses testing research) is: http://onlinelibrary.wiley.com/doi/10.1111/j.1469-8137.2012.04331.x/full

Example 2: Three tissues of the water fern Azolla filiculoides were RNA-seq'ed and unigenes created by assembly. No fern has been sequenced todate, neither as a genome, nor as a transcriptome. The sequencing was deep enough to be complete but, due to the phylogenetic distance, the fern has more unique transcripts that cannot be annotated compared to dicot plant transcriptome sequencing (80% annotation rate)

Test:

i) annotate the unigenes by Blast against A. thaliana, against S. moellendorffii and P. patens

ii) count the number of annotated and non-annotated transcripts, calculate the annotation percentage

iii) map the unigenes at the Kegg automatic annotation server and inspect the maps for completeness of central metabolism

Visualization:

i) Venn diagram of annotations

ii) map output from KAAS

Biological information from the assembly

You can check how many transcripts of a reference database you were able to assemble. Before you do that, you want to formulate a few hypotheses. You know most about your tissue or your mutant. Take the time to make these predictions and see if they come true. If they do, fine; if they don't, you have not yet really understood your tissue or mutant. As a simple example, if you sequence a leaf transcriptome, you expect that you will get contigs for all the genes involved in photosynthesis. You may also expect that genes involved in regulation are underrepresented because of the large dynamic range in leaves. How can you test hypotheses like these in a way that is publishable?

1. Produce a unigene list from the unigene database

Navigate to the folder containing the unigenes database:

```
$ grep ">" Azolla_unigenes.fasta > listofallunigenes.txt
```

2. Use a texteditor to remove the `>` from the dataset
3. For each sequence in the annotated reference, map whether it is covered by a contig or not.

   Extract the relevant information from a Blast Output File

```
$ cut -f2 ExampleBlastResult.txt | sort | uniq  >
  ExampleBlastResult_reduced.txt
```

   Open ExampleBlastResult_reduced.txt with a spreadsheet program. Add a "y" in a second column behind each entry.

4. Connect all datasets

   Copy the ExampleBlastResult_reduced.txt into the spreadsheet the list of unigenes. Make sure that the Identifiers in both sheets are exact matches (including capital letters). Use VLOOKUP (SVERWEIS in german) for string pattern matching: =VLOOKUP(Lookupvalue; SearchMatrix;column for return value;FALSE). Fill the column. Copy the column and reinsert values only.

5. Draw a Venn diagram (e.g. http://bioinfogp.cnb.csic.es/tools/venny/)

Example 3: Multiple tissues of Z. mays were RNA-seq'ed (Sekhon et al. 2012). Similar tissues have similar genes expressed, i.e. all leaves and all seed derived samples share gene expression patterns.

Test:

i) Load data into HCL (Hierarchical sample clustering) or PCA (principal component analysis) tool (see below point 7 & 8 for more information)

ii) cluster data to check whether similar tissues cluster together

Visualization:

i) Plot of the PCA and/or HCL of samples

Biological interpretation of more than two samples

Sometimes you are not only comparing two stages but you want to analyze a series of timepoints, tissues, species, etc that do NOT share a common reference point. You want to determine which genes have similar and which genes have different expression patterns to identify genes that may be involved in the same pathway or response. In this case, the same analyses strategies that have been applied to micro array analyses of the same type can be applied: hierarchical clustering, k-means clustering, weighted gene co expression network analysis, others. We will explore one out of many possible ways.

Open `ExampleFile_multiplesamples.txt` with a text editor. It is in the format: IDENTIFIER, DESCRIPTION, READ COUNT VALUE SAMPLE1, 2, 3, … Each sample is a log2 transformed rpkm value.

1. Open the MultiExperiment Viewer (MeV; http://www.tm4.org/mev/).
2. Load the data by following the instructions on your screen (ignore the possibility of annotations).
3. Transform the data by mean (or median) centering the rows.
4. Save the matrix with a new name.
5. Open a second MultiExperiment Viewer and load the new matrix. As a first step, we will inspect if the samples "look okay".
6. Under "Samples", color each group of replicate samples with a different color code.
7. Run a Principle component analysis (PCA): all replicates should group closely together, but apart from other samples in the different dimensions; similar tissues should cluster together

8. Run a hierarchical sample clustering (HCL) with the settings Euclidean, average linkage on the samples, NOT the genes: Replicate samples should group together; similar tissues should cluster together

These tests can also be applied to experiments with two samples, if each has multiple replicates, and the tests give a good indication if the sampling and sample processing went right. Now we will cluster the samples. First we will use k-means clustering. This clustering type forces you to specify how many clusters you want and the algorithm will sort them into clusters. You can theoretically use a Figure of Merit (FOM) analysis to determine the number of clusters statistically. In practice, FOM only works if the number of rows to be clustered is sufficiently small. Depending on the dataset, it could be about a hundred to a thousand. If you apply a FOM, the graph will flatten to 0 and where it does so, is the 'right' number of clusters. This never works with gene expression data, but still.

9. Run a FOM analysis. Specify that you want up to 20 clusters and three repeats of the analysis. Inspect the graph, you will not find the flat line, rather a constant decrease rate. (Doing a FOM will impress the statistical reviewer of your paper, results straight to supplemental, of course.)
10. Run a k-means clustering with a cluster number (say 12), and specify Euclidean or Pearson.
11. Inspect the results.
12. Members of each cluster can be exported as txt files (and added to an annotation and probed for pathway enrichment in particular clusters ☺). The pattern themselves might provide important clues about the dataset.

For hierarchical clustering, a lot of memory is required. The calculation time depends on the RAM available to the program.

SNP calling

CLC will call SNPs for you. It requires a read mapping prior to executing SNP calling.

1. Open CLC genomics workbench. Import a reference. Import reads.
2. Map the reads by choosing map reads to reference with default parameters.
3. Open the SNP detection tool. Choose the read mapping. CLC will offer options as to when to annotate a SNP (leave the defaults for quality, adjust the significance values based on your own parameters; e.g. require at least a ten read coverage and require that one variant can only be called a SNP if it is detected in at least 10% of the variants; values may vary based on your sample!)
4. CLC will create an exportable table.

If your RNA-seq sample is from a species without a sequenced reference genome, you can use this tool to determine the genetic variation in your species. To properly interpret your data, it is important to be aware of the number of individuals the RNA was isolated from.

Splice isoform differentiation

If mapping on a reference sequence with annotation (by gff file, see Gene Expression, CLC bio), CLC bio will return splice isoform mapping percentages immediately. Splice isoform discovery is possible with open source mapping tools but is not covered in this workshop.

Other – less obvious – pieces of information contained in the dataset

Your database contains expression values which are usually quite well compareable to qRT-PCR data (but there might be of course some sequencing biases). Nevertheless unlike microarray experiments, you can ask questions like “How much mRNA does the cell devote to one pathway?” Guess what percentage of the total transcriptome encodes genes of the electron transfer chain in the chloroplast? 10%? 30%? 50%? With the absolute values in hand we can look it up. Assume, you have sequenced a really unusual tissue, maybe nectary glands, glandular trichomes, epidermal cells of carnivorous plants. Which transcripts may dominate those tissues in absolute numbers? If you sequence a particular tissue in a particular species, it may help if you develop hypotheses about what you are likely to see BEFORE you ever look at the data. Those hypotheses will depend on the available literature and are a good starting point for writing up the paper at a later stage.

The expression values may also help you identify which isoform is active in a particular pathway. While not absolutely true (exemption example, sulfur assimilation), the mRNAs encoding enzymes in

a pathway, are reasonably similar in expression. The major isoforms responsible for primary metabolism can generally be identified from the data even it is not annotated as such in databases.

In our experience, metabolite flux roughly equals transcript abundance [The statement sounds crazy, we know]. But we can predict from a dataset with reasonable accuracy which pathways carry a high flux and which do not. We cannot make predictions about single enzymes since the maximal flux depends on kcat AND protein amount (and posttranslational modifications and probably half a dozen other things), but for pathways with multiple genes we can.

Biological interpretation of two samples

The analysis of RNA-seq data is similar to the analysis of microarray data. Consequently, similar analysis tools can be used.

In a publication focused exclusively or mainly on the RNA-seq analysis, only very limited discussion about single candidate genes will be tolerated by the reviewers. In our experience, about 80% of the manuscript will have to focus on the analyses at a larger scale. Given this focus, at the end of the manuscript, a little speculation will be okay. Of course if you extracted a single gene and showed by ko/ox that this particular gene carries a role, this might be a different story. Usually, however this will be a new manuscript

RNA-seq analysis can and will serve to identify candidate genes. In this case, a large part of the publication will focus on the results generated by this single gene analysis. We will now briefly discuss how to identify candidate genes from an experiment.

1. Produce an annotation for the reference sequences you are using. Think about publicly available information important for your project that you could add. Using the Linux commands, text editor tools and a spreadsheet program, almost all tabular information can be transferred. For now, use the annotation information produced earlier.
2. Map the read count data for all samples and replicates onto each reference sequence using VLOOKUPas you did for the statistical evaluation (→Statistical Evaluation of readcounts)
3. Also add e.g. MapMan or other annotations as you did in (→Biological information from the assembly) as the gene codes as such won't be very meaningful
4. Now you are able to play with the data, resorting according to different parameters and looking through the annotations whether a gene or a whole group of genes appears to be a good candidate for whatever you are seeking. This format can also be presented to other members of your team, who were not involved in the analysis so that they can look for interesting single genes. This is because most people know how to navigate in Excel

A table format like this one also enables you to check pathway behavior or check on a group of genes for which you have included a specific annotation, maybe your pathway of interest. The inspection of the table may suggest testable hypothesis for the large scale analysis.

Let's recall the methods:

1. If you have created the annotation table for single gene analysis, you can now use `COUNTIFS` functions to probe pathway enrichment
2. Start by establishing how many members are present in each category `=COUNTIF(target column;criterion)`
3. You might want to change p-values just to indicator flags significant (1) or non-significant (0), but this is of course up to your liking.
4. Calculate how many members of each category are present among your group (e.g. significant changed genes, upregulated); `=COUNTIFS(target column 1; criterion1; significance column; "y"; log change column; ">0")`
5. Calculate the number of references in your group (e.g. significant changed genes, upregulated) and the total number of references.
6. Install a Fishers Exact Test AddIn (i.e. http://www.obertfamily.com/software/fisherexact.html) or write a VBA script or use a webtool (http://www.quantitativeskills.com/sisa/statistics/fisher.htm) for Fishers Exact Test. Fishers Exact Test sort of works like a chi square but can be used safely with small numbers. The output is a p-value.
7. Calculate a Bonferroni correction (or any other correction you prefer) of the p-value based on the number of categories you tested. For Bonferroni the new alpha = alpha/number of categories
8. Draw conclusions

Having your own analysis pipeline enables you to get creative. If (for instance) you suspect a connection with auxin and gene expression changes; your testable hypothesis could be :"Auxin upregulated genes are overrepresented in the genes upregulated in mutant X compared to wildtype." You can analyze a publicly available microarray experiment (for example from AtGenExpress) with a standard pipeline for differentially expressed genes (we recommend RobiNA; http://mapman.gabipd.org/web/guest/robin) and map the 'significantly upregulated under auxin treatment' into the Annotation File using VLOOKUP functions. Afterwards, you can test enrichment with a Fishers Exact Test. No alpha correction required.

Or if you suspect a connection to any phytohormone… You guessed it; you analyze multiple publicly available microarray experiments with the standard pipeline and map the information into different columns. You test enrichment. This time you need to correct the alpha based on how many different hypotheses you tested.

Anything you can map into the annotation table can be tested for over- or underrepresentation! **WARNING** However there is one little problem with RNAseq (Here it is worse than microarrays). Your sensitivity for differential expression is not equal everywhere. Consider this: an upregulation from 2 to 4 reads is a 2 fold change, but so is an upregulation from 2000 to 4000. And indeed the statistical power for differential expression is higher in the latter. Therefore it is easier to detect differential expression for the second gene. Now if you do a Fisher test for enrichment of differentially expressed genes, you might miss a lot of the lowly expressed genes just because you couldn't detect differential expression (which doesn't mean it is not there). If there is a bias in functional groups between high and low expression, this bias will be carried over. E.g. genes for photosynthesis might be very high and transcription factors low. So in an experiment where processes change randomly you would think that photosystem is more effected than though by chance and transcription factors less than expected by chance.

Recommendation: R provides tools to correct for this bias, for an intial overview it is often enough to just use the Fisher test as described above.

You could also use GO terms for the enrichment analysis or display the group of genes on pathways with the help of the KAAS server or the MapMan server (http://mapman.mpimp-golm.mpg.de/general/ora/ora.shtml). Another possibility is to visualize the ratios and to analyze for pathway enrichment using a Fisher test in the MapMan GUI tool:

1. Download MapMan (http://mapman.gabipd.org, already available on the computer in this case).
2. Download the Arabidopsis annotation table (called mapping) from the MapMan store (Resources category; already installed on your computer).

Mapman requires three pieces of information to display data: the experiment, the mapping and the pathway. Mapping and pathway are already provided with Mapman, but you need to produce the experiment table.

1. Save your annotation table with the log fold change information to a new name.
2. Delete all columns except the identifier and the log fold change.

3. For Mapman to work, the identifier needs to have the exact same string syntax as the identifier in the mapping file. That means small a, small t, small g, no transcript suffix .X. Change the letters with the replace function. There is no transcript suffix. Save the table as a simple textfile.
4. Import the text file as a new experiment.

Click through the different pathways and note what has changed.

It is possible to carry out the gene expression analysis in a Java based tool, RobiNA (http://mapman.gabipd.org/web/guest/robin). Essentially the tool combines the read mapping, the statistical analysis and the visualization into one workflow. The advantage is a GUI, no need of the command line. The disadvantage is that you can only do what the developers envisioned. RobiNA is installed on the computer. Have a try if you have time or try it at home.

Annex 1: Setting up your environment

Setup your working environment - Examples

1 Installation of BLAST

i) Download the recent BLAST release for Linux 64Bit from ftp://ftp.ncbi.nlm.nih.gov/blast/executables/release/LATEST

ii) The zipped file contains executables, which don't need to be installed, they just have to be extracted and copied to the correct directory (here: ~/software)

Change to your home directory and create a directory for software:

```
113$ cd
114$ mkdir software
```

Change to 'Downloads' directory (we assume that this is the file was downloaded to):

```
115$ cd ~/Downloads
```

To uncompress the files, use the `untar` command with the flags `-xvzf`:

```
116$ tar -xvzf blast-2.2.26-x64-linux.tar.gz
```

Move the contents to your software directory:

```
117$ mv -n blast-2.2.26/* ~/software/ -r
```

Test if installation is successful:

```
118$ blastall
```

2 Installation of BLAT

i) Download the BLATsuite from Jim Kent's Website http://hgwdev.cse.ucsc.edu/~kent/exe/linux/

If you want the 64 bit version of BLAT, you can find this on the following site:- http://hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64/blat/

ii) The most recent version is v36: `blatSuite.36.zip`

iii) move the .zip file to you software directory and unzip it

Change to the 'Downloads' directory and move blatSuite36.zip to the software directory:

```
119$ cd ~/Downloads
120$ mv blatSuite36.zip ~/software/
```

If in doubt, whether to use `unzip` or `gunzip` you can always check what Unix suggests by using the `file` command.

```
121$ file blatSuite36.zip
```

Now uncompress the zip file using unzip:

```
122$ unzip blatSuite36.zip
```

Test if BLAT is working:

```
123$ ~/software/blat
```

3 Installation of FASTX

a. Go to the Galaxy-FASTX toolkit download page here: http://hannonlab.cshl.edu/fastx_toolkit/download.html

b. Download the latest version of fastx_toolkit and libgtextutils and save to Downloads

c. Open terminal

Change directory to the `Downloads` directory:

```
124$ cd Downloads
```

Extract the downloaded packages, e.g. `libgtextutils-0.6.tar.gz` and `fastx_toolkit-0.0.13.tar.bz2`

When the file suffix is `.tar.bz2`, one needs to provide the `-j` parameter to tar

```
125$ tar -xzvf libgtextutils-0.7.tar.gz
126$ tar -xvjf fastx_toolkit-0.0.14.tar.bz2
```

Install gtextutils:

```
127$ cd libgtextutils-0.7
128$ ./configure --prefix=/home/$USER/software
129$ make
130$ make install
```

Install fastx-toolkit:

```
131$ export
   PKG_CONFIG_PATH=/home/$USER/software/lib/pkgconfig:$PKG_CONFIG_PATH
132$ cd ~/Downloads
133$ cd fastx_toolkit-0.0.14
134$ ./configure --prefix=/home/$USER/software
135$ make
136$ make install
```

The compiled programs can now be found in your home directory in `software/bin/`

4 Installation of Trinity

i) Download the current Trinity package from https://github.com/trinityrnaseq/trinityrnaseq/releases

ii) Move, unpack, and compile Trinity source code

```
137$ mv trinityrnaseq-2.0.6.zip ~/software
138$ cd ~/software
139$ unzip trinityrnaseq-2.0.6.zip
140$ cd trinityrnaseq-2.0.6
141$ make
```

5 Finding sequence files for reference genome (example TAIR10 coding sequences fasta file)

i) Open google.com in your browser

ii) Search for terms such as “Arabidopsis” “genome” “blast” “blast database” “blast set”

iii) In almost any possible combination you will be referred to www.arabidopsis.org

iv) Since you want to Download the geneset, instead of using it online, click on “Download”

v) This redirects you to the FTP-section of the website. Here you can browse similar to the standard file explorer.

vi) There are several ways to find the desired dataset, as there are crosslinks throughout the directory tree. The most intuitive ones are:

i. Sequences->blast_datasets/TAIR10_blastsets/… (<-newest version)

ii. Genes->TAIR10_genome_release/…->TAIR10_blastsets

1. The file you need is: TAIR10_cds_20101214_updated

vii) Right Click -> Save As/Save Link as -> Select folder

6 Installation of bowtie/tophat/cufflinks

i) Go to respective sites and down load binaries

ii) for bowtie: http://bowtie-bio.sourceforge.net/bowtie2/index.shtml, click on 'latest release' and then the linux version

iii) for tophat: http://ccb.jhu.edu/software/tophat/index.shtml, click on the linux binary under 'Releases'

iv) for cufflinks http://cole-trapnell-lab.github.io/cufflinks/install/, click on the latest linux release

v) Move to software directory and unzip.

```
142$ cd ~/Downloads
```

Move to `~/software/` (exact version names may change as programs are updated)

```
143$ mv bowtie2-2.2.5-linux-x86_64.zip ~/software/
144$ mv tophat-2.0.13.Linux_x86_64.tar.gz ~/software/
145$ mv cufflinks-2.2.1.Linux_x86_64.tar.gz ~/software/
```

Change to `~/software` and extract files:

```
146$ cd ~/software
147$ unzip bowtie2-2.2.5-linux-x86_64.zip
148$ tar -zxf tophat-2.0.13.Linux_x86_64.tar.gz
149$ tar -zxf cufflinks-2.2.1.Linux_x86_64.tar.gz
```

Test whether the programs work:

```
150$ cd bowtie2-2.2.5
151$ ./bowtie2
152$ cd ../tophat-2.0.13.Linux_x86_64/
153$ ./tophat2
154$ cd ../cufflinks-2.2.1.Linux_x86_64/
155$ ./cufflinks
```

.....

.....

.....

.....

R: Installation of R

If you run a Linux Operating System, R usually is delivered along with it. However in case it is not, or you need a different version of it, here is a very short "how to install", requiring you to be admin on the computer.

Attention: If you are not the admin at this computer, make your admin install the R Package for you. R is no fun - if it works at all - when installed in a reduced-permission user environment.

1. The golden way to install R in a Linux environment is using the native package manager.

Install R on Ubuntu Linux:

```
156$ sudo apt-get install r-core
```

Install R on Fedora Linux:

```
157$ sudo yum install R-core
```

2. An installation into Mac OSX or Windows environment is much easier. Just download and double-click the corresponding installer from http://cran.r-project.org/

3. After installing the R base software, the computer is able to understand R scripts, but due to the nature of R, this just works at command-line level. For user-friendly working with R we suggest the use of a GUI. Go to http://rstudio.org/

4. RStudio is available for all three kinds of operating systems. The installation is as easy as downloading and double-clicking.

5. If you open RStudio for the first time, you might be asked for the path R was installed to (not if you used a package manager under Linux)

6. Click on New R Script or open one you downloaded somewhere.

7. Now you see that RStudio is usually divided into four views. Top-Left is the Scripting area that is where the Script is shown or typed into. There are buttons to execute the script, either line by line or the whole script at once. Top-Right is a Workspace Overview that shows the names and contents of all variables that your R session knows at that time. You don't need to understand the concept of variables in R, but this view is nice to see what R actually does with your data. You can click double click on any variable to get a primer on what the contents look like. Bottom-Left you see the R console. As R is a mathematical (programming) language, the simplest form of an interactive parser, that tells the computer what to do, but still interacts with the user, is a console. You can use it to type R commands and execute them; however, we would suggest using the scripting area because it is easier to track commands and save as file after executing. Finally the Bottom-Right area is where the plots/figures are drawn. There are buttons to switch between multiple plots (arrows) and to export the graphics to a file.

8. To give a good example of very complex code, that is easy to handle as a biologist, we will run you through the installation and execution of the "DEGSeq" package that was developed to detect differential gene expression in RNA-seq datasets.

   a. Visit http://www.bioconductor.org/packages/2.6/bioc/html/DEGseq.html

   b. On the landing page, there are already instructions to install the package, follow them

c. In your Scripting area (Click File New->New R Script if necessary) type:

```
source("http://bioconductor.org/biocLite.R")
biocLite("DEGseq")
```

d. Select the two lines with your cursor and click on “run” which runs the current line or the selection (if available). Watch the console.

e. Run the first few lines of the example code from the “Example for DEGSeq” in the PDF manual.

```
kidneyR1L1 <- system.file("extdata", "kidneyChr21.bed.txt", package =
"DEGseq")
liverR1L2 <- system.file("extdata", "liverChr21.bed.txt", package =
"DEGseq")
refFlat <- system.file("extdata", "refFlatChr21.txt", package =
"DEGseq")
mapResultBatch1 <- c(kidneyR1L1)
mapResultBatch2 <- c(liverR1L2)
outputDir <- file.path(tempdir(), "DEGseqExample")
```

f. What you should have noticed by now, the R console is always preceded by a “>” symbol. This is to tell the user, that R is expecting input.

g. The next command in the example is a multiline command. The second line of the command is preceded by a + sign. This tells the user, that the line is only continuing a preceding line and is not a new command.

```
> DEGseq(mapResultBatch1, mapResultBatch2, fileFormat = "bed",
+     refFlat = refFlat, outputDir = outputDir, method = "LRT")
```

Attention: If you copy the above command into R, you will get an error. Why is that, you just copy&pasted??? The reason is, that in some PDFs not the R scripts are printed, but the console output when the script is executed. That results in commands being extended by preceding “>” and “+” signs. If you remove them the code will work.

h. We will further use this package in the gene expression section.

Copyright

Legal code of the license:

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(b), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(b), as requested.

b. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4 (b) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

c. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.