GETTING STARTED

The data files required for this workshop are located in a public directory of the instructor's account. You need to have this in your home directory (/home/username) before you start the exercise 1. You can download the compressed directory and extract it by simple commands given below. You will learn about these commands later in the exercise.

Open the terminal and enter these commands (commands are case sensitive) and each command should be entered in a single line followed by ↵ (Enter) key

```
wget http://www.public.iastate.edu/~arnstrm/WORKSHOP_FILES.tar.gz ↵
```

```
tar –xvzf WORKSHOP_FILES.tar.gz ↵
```

Once your cursor (command prompt) comes back to the original position, type

```
ls ↵
```

You should see `WORKSHOP_FILES` listed there.

PS: hand-outs/files are also available for download at https://github.com/ISUgenomics/Basic_UNIX

UNIX EXERCISE 1

This exercise is designed to provide the basic skills required for working in the UNIX environment, using plenty of relevant examples, specifically for biologists. If you are using your personal computer, make sure that you have downloaded the files required for the workshop. This exercise will provide you information regarding navigation, files and directory creation/modification and some administrative things related to file permissions.

NAVIAGATION

This section will introduce you to some basic file/directory navigation and manipulation techniques.

TO KNOW THE PRESENT LOCATION OF YOUR COMMAND

`pwd`

`/home/username`

Returns you the present working directory (*p*rint *w*orking *d*irectory)

This means, you are now working in the `username` directory, which is located in `home` directory. The directory that you will be in after logging in is your home directory. You can also avoid writing the full path by using `~` in front of your `username` or simply `~`.

`~ or ~username` same as `/home/username`

Present directory is represented as `.` (dot) and parent directory is represented as `..` (dot dot)

CHANGING DIRECTORIES

To jump from one directory to another we use the `cd` (*c*hange *d*irectory) command.

`cd ..`
Changes your present location to the parent directory

`cd DIRECTORY`
This changes your location back to your DIRECTORY.

Task 1.1: Now change your directory to the `WORKSHOP_FILES` directory present in your home directory.

NOTE: You can type in first few letters of the directory name and then press `tab` to auto complete rest of the name (especially useful when the file/directory name is long). This only works when there are unique matches for the starting letters you have typed. If there is more than one matching files/directories, pressing `tab` twice will list all the matching names. You can also recall your previous commands by pressing `up/down arrow` or browse all your previously used commands by typing `history` on your terminal (typically, last 500 commands will be saved in this file).

DIRECTORIES AND FILES

MAKING DIRECTORIES

To create a directory, `mkdir` (*m*a*k*e *dir*ectory) can be used.

```
mkdir DIRECTORY
```

Unlike PC/Mac folders, here you can't have space in your directory name (but some special characters are okay). You can also specify the path where you want to create your new folder.

Task 1.2: Make a new directory named `FirstDirectory` within the `WORKSHOP_FILES` directory. Then change your directory to the `FirstDirectory`.

```
mkdir FirstDirectory
```

COPYING DIRECTORIES

To copy a file, `cp` (*cop*y) command is used. When using this command you have to provide both source file and destination file.

```
cp SOURCE DESTINATION
```

You can also specify the absolute path of the source and/or destination file. To know more about any command you can use `man` command, which opens the manual of the command you ask (referred as 'man page').

```
man cp
```

This opens the manual for the `cp` command. Take a look at the manual of `cp` command (use arrow keys to move top or bottom of the page). `OPTIONS` are optional arguments that can be used to accomplish more from the same command. *Eg.,* by using option `-i` with the regular `cp` command, you can always make sure that you are not overwriting the existing file while copying. The syntax for using the options will also be provided in the manual. **To exit, press `q`.**

Looking at the man page for cp command, what options can be used to copy a directory (including all files within it)? _____

How else you can get help on cp command (other than 'man')? _____

Task 1.3: Now change your directory back to the home directory. Create a copy of `WORKSHOP_FILES` and name it as `BACKUP_WORKSHOP`). This will serve as a backup copy of all files that are required for the workshop (in case you accidentally modify the contents while working).

```
cp -r WORKSHOP_FILES BACKUP_WORKSHOP
```

MOVING DIRECTORIES

To move a file or a directory, `mv` (*mo*ve) command is used. Again, like the `cp` command you need to provide both source file and destination file.

```
mv SOURCE DESTINATION
```

Absolute path also works fine. Some of the options used by `cp` command also work with `mv` command. `mv` can also be used to rename files and directories

```
mv OLDNAME NEWNAME
```

Task 1.4: Rename `WORKSHOP_FILES` as `tutorials`.

```
mv WORKSHOP_FILES tutorials
```

VIEWING THE CONTENTS OF THE DIRECTORY

The contents of a directory can be viewed using `ls` (*ls*t) command.

```
ls DIRECTORY                # now try it with tutorials directory
```

If no directory name is provided then `ls` will list all the contents of the present directory.

Like any other command, you can use absolute path or abbreviated path. There are also various options available for `ls` command.

Some very useful options include:

`ls -l`
Lists all the files in lengthy or detailed view

`ls -t`
Lists all the files, sorted based on creation time

`ls -S`
Lists all the files, sorted based on size

You can also combine these options together for getting more focused results.

Looking at the manual for ls, what option can you use to view hidden files in a directory (files starting with dot)? _____

Can you sort the files based on its extension? How? _____

Task 1.5: Examine the contents of the `tutorials` directory. Try options such as `-l`, `-t`, `-a` and `-X`. Also check if you can combine many options together (like `-la` or `-lh` etc). Try these:

```
ls -l tutorials
ls -a
```

```
ls -1 tutorials
ls -lh tutorials
ls -t tutorials
```

CREATING AND EDITING FILES

`touch FILENAME`
Creates a new file in the present location

`nano FILENAME`
Like notepad/textedit, this text editor lets you edit a file.

Task 1.6: Create a new file named `firstfile` inside the `tutorials` directory. You can create using touch or using `nano`. Then add some contents (Your name and email address) to the `firstfile` (using `nano`). After editing, press `Ctrl + X` to exit, then enter `y` to save changes and confirm the file name.

```
touch firstfile
nano firstfile
```

VIEWING CONTENTS OF THE FILES

There are various commands to print the contents of the file in bash. Most of these commands are often used in specific contexts. All these commands when executed with filenames displays the contents on the screen. Most common ones are `less`, `more`, `cat`, `head` and `tail`.

`less FILENAME` *try this:* `less AT_cDNA.fa`
Displays file contents on the screen with line scrolling (to scroll you can use `arrow` keys, `PgUp/PgDn` keys, `space bar` or `Enter` key). **When you are done press `q` to exit**.

`more FILENAME` *try this:* `more AT_cDNA.fa`
Like less command, also, displays file contents on the screen with line scrolling but uses only `space bar` or `Enter` key to scroll. **When you are done press `q` to exit**.

`cat FILENAME` *try this:* `cat AT_cDNA.fa`
Simplest form of displaying contents. It *cat*alogs the entire contents of the file on the screen. In case of large files, entire file will scroll on the screen without pausing

`head FILENAME` *try this:* `head AT_cDNA.fa`
Displays only the starting lines of a file. The default is first ten lines. But, any number of lines can be displayed using `-n` option (followed by required number of lines).

`tail FILENAME` *try this:* `tail AT_cDNA.fa`
Similar to head, but displays the last 10 lines. Again `-n` option can be used to change this.

More information about any of these commands can be found in `man` pages (`man` command)

Task 1.7: Try using all these commands on the `RefSeq.faa`. You are also welcome to try these commands on various other files that are present in the `tutorials` directory. These commands don't change the contents of the file; they just display them on the screen.

DELETING FILES AND DIRECTORIES

To delete directories from the system, you can use `rmdir` (*rem*ove *dir*ectory) command. You can also use `rm` command to delete file(s).

`rmdir DIRECTORY`
The directory should be empty before you use the `rmdir` command.

`rm FILE`
To delete a file `rm` command can be used

Some useful options include
`-r` recursively delete files
`-f` delete forcefully

`rm -rf DIRECTORY` **[DO NOT USE THIS NOW!]**
When you want to delete a folder, with all its content

Task 1.8: Delete the directory named `delete_me` inside the `tutorials` directory (to do this you may first want to delete the `sample.txt` file inside this directory).

```
cd delete_me
rm sample.txt
cd ..
rmdir delete_me
```

COMPRESSING FILES

There are several options for archiving and compressing groups of files or directories. Compressed files are not only easier to handle (copy/move) but also occupy less size on the disk (less than 1/3 of the original size). In Linux systems you can use `zip`, `tar` or `gz` for archiving and compressing files/directories.

ZIP compression/extraction

`zip OUTFILE.zip INFILE.txt`
Compress `INFILE.txt`

`zip -r OUTDIR.zip DIRECTORY`
Compress all files in a `DIRECTORY` into one archive file (`OUTDIR.zip`)

`zip -r OUTFILE.zip . -i *.txt`
Compress all txt files in a `DIRECTORY` into one archive file (`OUTFILE.zip`)

`unzip SOMEFILE.zip`

Decompress a file

Task 1.9: Zip `AT_genes.gff` file located in the `tutorials` directory. Check the file size before and after zip compression (Hint: use `ls -lh` to check file sizes).

```
zip AT_genes.gff.zip AT_genes.gff
```

Is there any size difference before and after compressing? *Y/N*

`tar` (*t*ape *ar*chive) utility saves many files together into a single archive file, and restores individual files from the archive. It also includes automatic archive compression/decompression options and special features for incremental and full backups.

`tar -cvf OUTFILE.tar INFILE`
archive `INFILE`

`tar -czvf OUTFILE.tar.gz INFILE`
archive and compress file `INFILE`

`tar -tvf SOMEFILE.tar`
list contents of archive `SOMEFILE.tar`

`tar -xvf SOMEFILE.tar`
extract contents of `SOMEFILE.tar`

`tar -xzvf SOMEFILE.tar.gz`
extract contents of gzipped archive `SOMEFILE.tar.gz`

`tar -czvf OUTFILE.tar.gz DIRECTORY`
archive and compress all files in a directory into one archive file

`tar -czvf OUTFILE.tar.gz *.txt`
archive and compress all ".txt" files in current directory into one archive file

Task 1.10: Archive and compress the `BACKUP_WORKSHOP` directory you created in Task 1.3 (you can name it as `backup.tar.gz` or anything you want)

```
tar -czvf backup.tar.gz BACKUP_WORKSHOP
```

`gzip` (*g*nu *zip*) compression utility designed as a replacement for compress, with much better compression and no patented algorithms. The standard compression system for all GNU software.

`gzip SOMEFILE`
compress `SOMEFILE` (also removes uncompressed file)

`gunzip SOMEFILE.gz`
uncompress `SOMEFILE.gz` (also removes compressed file)

Task 1.11: `gzip` the file `AT_genes.gff` and examine the size. `gunzip` it back so that you can use this file for the later exercises.

```
gzip AT_genes.gff
ls -lh
gunzip AT_genes.gff.gz
ls –lh
```

ADMINISTRATIVE COMMANDS

CHANGING PERMISSIONS

All files in the UNIX system will have a set of permissions which define what can be done with that file and by whom. (What = read (view contents), write (modify) and execute (run script) Whom=User (owner), group (that account belongs to) and everyone else). They are denoted as

PERMISSIONS		**RELATIONS**	
read	r	owner	u
write	w	group	g
execute	x	others	o
		all users	a

To look at the permissions for any file, you can list the files with l option (`ls –l`).

```
Permissions User  Group Size  Date modified     Name
lrwxrwxrwx 1 arnstrm GIF    24 Jan  7 09:40 arnstrm -> /data006c/GIF_2c/arnstrm
drwxrwx--- 3 arnstrm GIF  4096 Jun  4 15:27 bin
drwxrwxr-x 5 arnstrm GIF  4096 Mar 18 09:10 coreutils
-rwxr-xr-x 1 arnstrm GIF 11908 Jan  7 13:07 cshrc_severin
drwxrwxr-x 4 arnstrm GIF  4096 Mar 18 09:17 dos2unix
-rw-rw-r-- 1 arnstrm GIF 46470 May 19 09:48 gtf2gff3.pl
drwxrwxr-x 4 arnstrm GIF  4096 Apr 10 09:15 igv
-rw-rw-r-- 1 arnstrm GIF   930 May 16 11:05 module_file.txt
-rwxrwx--- 1 arnstrm GIF  1228 Jun  5 14:51 template.sub
-rw-rw-r-- 1 arnstrm GIF 11326 May 19 09:47 validate_features.pl
  u   g   o
```

(d=directory, l=link, r=read, w=write, x=execute, -=blank, u=user, g=group, o=others)

To set/modify a file's permissions you need to use the `chmod` command (*ch*ange *mod*e). Only the owner of a file can alter a file's permissions. The syntax:

```
chmod [OPTIONS] RELATIONS[+ or -]PERMISSIONS FILE
```

Add permissions

`chmod RELATIONS+PERMISSIONS FILENAME`	
`chmod g+rwx FILENAME`	grants read, write and execute permissions for group
`chmod g+r FILENAME`	grants read permission for group
`chmod a+rwx FILENAME`	makes the file public (don't do this to any file/directory unless you want to share)

Remove permissions

`chmod RELATIONS-PERMISSIONS FILENAME`

`chmod g-wx FILENAME`	removes write and execute permissions for group
`chmod g-rwx FILENAME`	removes all permissions for group
`chmod a-rwx FILENAME`	removes all permissions for others
`chmod a-x FILENAME`	removes execution permissions for others

OPTIONS include

`-R`	recursively (the permissions are applied to all the files, directories present inside the directory)

Task 1.12: Check the permissions for the files located in the `tutorials` directory. Do

```
ls -l
```

What permissions does the group have on these files? _____

Which group does your account belong to? _____

UNIX EXERCISE 2

Second exercise on UNIX deals with more complex commands with their useful options and using multiple commands at a time. Make sure you understand all the commands from the previous exercise as you will be using them frequently in this exercise.

FASTA FORMAT:

`FASTA` format is nothing but a simple text file containing either nucleotide sequences or protein sequences. An individual sequence always starts with a single line description of the sequence, followed by lines of sequence data. Description can be just an identification number or even blank (not recommended) but should always begin with a greater-than (`>`) symbol. The sequence is considered to be complete if another line starting with a `>` is encountered. The simplicity of `FASTA` format makes it easy to manipulate and parse sequences using text-processing tools and any scripting languages like Python, Perl or Ruby.

Some example `FASTA` format protein sequences are given below:

```
>gi|18403023|ref|NP_565747.1| splicing factor 3A subunit 2 [Arabidopsis
thaliana]
MDREWGSKPGSGGAASGQNEAIDRRERLRRLALETIDLAKDPYFMRNHLGSYECKLCLTLHNNEGNYLAH
TQGKRHQTNLAKRAAREAKDAPTKPQPLKRNVSVRRTVKIGRPGYRVTKQYDPELQQRSLLFQIEYPEIE
DNIKPRHRFMSSYEQKVQPYDKSYQYLLFAAEPYEIIAFKVPSTEVDKSTPKFFSHWDPDSKMFTLQVYF
KPTKPEPNKPQSAVGANGLPPPPPPPPHQAQPPPPPPSGLFPPPPPPMANNGFRPMPPAGGFGHPNM

>gi|224140247|ref|XP_002323495.1| predicted protein [Populus trichocarpa]
MDREWGSKPGSGGAASAQNEAIDRRERLRRLALETIDLAKDPYFMRNHLGSYECKLCLTLHNNEGNYLAH
TQGKRHQTNLAKRAAREAKDAPALPQPNKRKVNIRKTVKIGRPGYRVTKQFDPETKQRSLLFQIEYPEIE
DNTKPRHRFMSSYEQRIEANDKRFQYLLFSAEPYEIIAFKVPSTEIDKSTPKFFSHWDPDSKMFTLQLYF
KLKPPEANKPQSVAAANSTVPSQPPPPLPPQGLPAGSRPPPPPMPASLPPPPPPAMANGPRPMPPGGAPP
APPPPPGGSGAMVNFTPGTQAGRPSSMLPPHGFLGQQMQGQTIRPPLLPPNMGQ
```

PIPES AND REDIRECTS

Many `UNIX` commands use some input file/data and display the output on the screen. This is feasible when the data being displayed is small enough to fit the screen or if it is the endpoint of your analysis. But for large data outputs, it is efficient to redirect to a file instead of screen. This can be done very easily in `UNIX` using `>` (greater than) or `<` (lesser than) or `>>` signs.

- `<` redirects the data to the command for processing
- `>` redirects the data from the command's output to a file. The file will be created if it is non-existing and if present it will overwrite the contents with the new output data (you will lose the original file).
- `>>` unlike `>` this redirection lets user append the data to an already existing file or a new file

- Another special operator | (called pipe) is used sometimes to pass the output from a command to another command (as input) before sending it to an output file or display.

Some *eg.*

`cat FILE1 > FILE2`
Creates a new file, `FILE2` with same contents as old file, `FILE1`

`cat FILE1 >> FILE2`
Appends the contents for `FILE1` to `FILE2`, equivalent to opening `FILE1`, copying all the contents, pasting the copied contents to the end of the `FILE2` and saving it!

`cat FILE1 | less`
Here, cat command displays the contents of the `FILE1`, but instead of sending it to standard output (screen) it sends it through the pipe to the next command 'less' so that contents of the file are now displayed on the screen with line scrolling.

Task 2.1: The `Sequences` directory contains a number of files and each of these files contain a single `FASTA` formatted nucleotide sequence. Combine them all together to make a single file `sequences.fasta` using redirects.

```
cat *.fa >> sequences.fasta
```
this command will combine all `.fa` files into one.

REGULAR EXPRESSIONS

When working with the sequences (protein or DNA) we are often interested to see if a particular feature is present or not. This could be various things like a start codon, restriction site or even a motif. In `UNIX` all strings of text that follow some pattern can be searched using some formula called regular expressions. *eg.* If you are looking for a particular motif in large number of sequences, then you can create a regular expression in `UNIX` and search all the sequences having that motif relatively easily. Regular expression consists of normal and metacharacters. Commonly used characters include

Expression	Function
.	matches any single character
$	matches the end of a line
^	matches the beginning of a line
*	matches one or more character
\	quoting character, treat the next character followed by this as an ordinary character.
[]	matches one or more characters between the brackets
[range]	match any character in the range
[^range]	match any character except those in the range
\{N\}	match N occurrences of the character preceding (sometimes simply +N) where N is a number.
\{N1,N2\}	match at least N1 occurrences of the character preceding but not more than N1
?	match 1 occurrence of the character preceding
|	match 2 conditions together, \(this|that)\ *matches both this or that in the text*

For complete list, type `info regex` on your terminal.

Some examples related to nucleotide/protein sequences:

Patterns	Matches
`^ATG`	Find a pattern starting with ATG
`TAG$`	Find a pattern ending with TAG
`^A[TGC]G`	Find patterns matching either ATG, AGG or ACG
`TA[GA]$`	Find patterns matching either TAG or TAA
`^A[TGC]G*TGTGAACT*TA[GA]$`	Find gene containing a specific motif
`[YXN][MPR]_[0-9]\{4,9\}`	Find patterns matching NCBI RefSeq (*eg* XM_012345)
`\(NP|XP\)_[0-9]\{4,9\}`	Find patterns matching NCBI RefSeq proteins

Some common commands that can be used to manipulate text using regular expressions are `grep` (filters input against a pattern), `sed` (applies transformation after searching a pattern) and `awk` (manipulates data arranged in columns). We will discuss these commands in detail

GREP

`grep` (*g*lobally search a *r*egular *e*xpression and *p*rint) is one of the most useful commands in UNIX and it is commonly used to filter a file/input, line by line, against a pattern e*g.,* to print each line of a file which contains a match for pattern.

```
grep PATTERN FILENAME
```

Like any other command there are various options available for this command. Most useful options include:

`-v`	inverts the match or finds lines NOT containing the pattern.
`--color`	colors the matched text for easy visualization
`-F`	interprets the pattern as a literal string.
`-H, -h`	print, don't print the matched filename
`-i`	ignore case for the pattern matching.
`-l`	lists the file names containing the pattern (instead of match).
`-n`	prints the line number containing the pattern (instead of match).
`-c`	counts the number of matches for a pattern
`-w`	forces the pattern to match an entire word.
`-x`	forces patterns to match the whole line.

With options, syntax is

```
grep [OPTIONS] PATTERN FILENAME
```

Some typical scenarios to use grep:

- Counting number of sequences in a multi-fasta sequence file
- Get the header lines of fasta sequence file
- Find a matching motif in a sequence file
- Find restriction sites in sequence(s)
- Get all the Gene IDs from a multi-fasta sequence files and many more.

Now let's use `grep` command to do some simple jobs with the sequences:

Counting sequences: By `FASTA` format definition, we know that number of sequences in a file should be equal to the number of description lines. So by counting `>` in file, you can count the number of sequences. This can be done using counting option of the `grep` (`-c`).

```
grep -c ">" FILENAME
```

Task 2.3: Count the number of sequences `AT_cDNA.fa` and `RefSeq.faa`

```
grep -c ">" AT_cDNA.fa          _________
grep -c ">" RefSeq.faa          _________
```

If you are looking for information about the sequences, you can list all the headers (description lines) for the sequences using `grep`. Simply search for `">"` and `grep` will list all the description lines.

```
grep ">" FILENAME
```

```
grep ">" AT_cDNA.fa
```

Alternatively, you can send it to a file if you want to use it later or you can just pipe it to `less` or `more` command to scroll through it line by line or page by page.

```
grep ">" FILENAME > HEADERFILE.txt
```

```
grep ">" FILENAME | less
```

```
grep ">" AT_cDNA.fa | less
```

Use `up/down arrow` keys to move up and down, press `q` to exit

See what kind of sequences are in `AT_cDNA.fa file`. Do they all seem to belong to same organism?
yes/no *Which organism?* _____

Using `grep` you can also locate all the lines that contain a specific term you are looking for. This is very useful especially to look for a specific gene among a large number of annotated sequences.

```
grep "word or phrase to search" FILENAME
```

Task 2.4: Try searching for your favorite gene, to see if it is present in `AT_cDNA.fa` (this file contains all annotated sequences for *Arabidopsis thaliana*). Unlike Google or any search engines, only exact search terms will be identified, but you can ask `grep` to ignore cases while searching using `-i` option. Try these:

```
grep -i "transcription factor" AT_cDNA.fa
```

```
grep -i "TFIIIA" AT_cDNA.fa
```

You can also use this feature to see if your sequence of interest has a specific feature (restriction site, motif etc.,) or not. This can be performed better using `--color` option of the grep.

Go to the sequences directory, search for *Eco*R1 (GAATTC) site in the `NT21.fa` file, and use the color option. Also, try looking for a C2H2 zinc finger motif in `RefSeq.faa` file (for simplicity let's assume zinc finger motif to be CXXXCXXXXXXXXXHXXXH. Either you can use dots to represent any amino acids or use complex regular expressions to come up with a more representative pattern. Try these:

```
grep --color "GAATTC" ./Sequences/NT21.fa
grep --color "C..C............H...H" RefSeq.faa
             (2)      (12)     (3)
```

You can also use `grep` command to exclude the results containing your search term. Say if you want to look at genes that are not located in chromosome 1, you can exclude it form your search by specifying `-v` option.

```
grep -i "transcription factor" AT_cDNA.fa| grep -v "chr1"
grep -i "transcription factor" AT_cDNA.fa| grep "chr1"
```

Notice the difference in output from the above two commands.

Try to understand the following command lines (and record your results, where applicable):

```
grep -c -w "ATP" RefSeq.faa                    _____________
grep -c CGT[CA]GTG AT_cDNA.fa                  _____________
grep -l "ATG" ./sequences/*.fa
```

You can also try some regular expressions related to nucleotide/protein sequences provided earlier to see how it works.

SED

The `sed` command is a stream *ed*itor that reads one or more text files, makes changes or edits according to editing script, and writes the results to standard output. Most common editing script `sed` uses is to substitute text matching a pattern. The simple syntax for using `sed` is as follows

```
sed 'OPERATION/REGEXP/REPLACEMENT/FLAGS' FILENAME
```

Here, `/` is the delimiter (you can also use `_` (underscore), `|` (pipe) or `:` (colon) as delimiter as well)

`OPERATION` specifies the action to be performed (sometimes if a condition is satisfied). The most common and widely used operation is `s` which does the substitution operation (other useful operators include `y` for transformation, `i` for insertion, `d` for deletion *etc.*).

`REGEXP` and `REPLACEMENT` specify search term and the substitution term respectively for the operation that is being performed.

FLAGS are additional parameters that control the operation. Some common FLAGS include:

g	replace all the instances of `REGEXP` with `REPLACEMENT` (globally)
n	(n=any number) replace nth instance of the `REGEXP` with `REPLACEMENT`
p	If substitution was made, then prints the new pattern space
i	ignores case for matching `REGEXP`
w	file If substitution was made, write out the result to the given file
d	when specified without `REPLACEMENT`, deletes the found `REGEXP`

For brevity we only discuss `sed` command with respect to search and replace function. To do other things please refer to the man page of sed or the reference provided here http://www.grymoire.com/Unix/Sed.html#uh-47.

Some search and replace examples:
`sed 's/chr/chromosome/g' FILENAME` replaces ALL instances in a line
`sed '/MTF1/s/chr/chromosome/g' FILENAME`
replaces all instances in a line only if it contains 'MTF1'

Other common tasks that can be performed using `sed`

`sed -n '52p' FILENAME`	Prints 52nd line
`sed -n '8,12p' FILENAME`	Prints line 8 through 12
`sed -n '1,2~2p'  FILENAME.fastq`	Prints 2nd and every 4th line (header and sequence from a FASTQ file)
`sed "1d" FILENAME`	Delete 1st line
`sed "1,3d" FILENAME`	Delelte line 1, 2 and 3
`sed 's/^$//g' FILENAME`	Delete balnk lines
`sed '2 i line to insert' FILENAME;`	insert "line to insert" on second line

Task 2.5: Try using replace function on `AT_genes.gff` file (to change `Chr` to `Chromosome`). View both files to see the difference.

```
sed 's/^Chr/Chromosome_/g' AT_genes.gff > AT_genes_converted.gff
```

AWK

Unlike other UNIX commands `awk` is a structured language by itself. `awk` stands for the names of its authors *A*ho, *W*einberger, and *K*ernighan. Many bioinformatics programs generate rows and columns of information. `awk` is an excellent tool for processing these rows and columns, and it is easier than most conventional programming languages.

The typical syntax for `awk` is:

```
awk 'PATTERN {ACTION}' FILENAME
```

`awk` then works by reading the input file one line at a time, matching the given `PATTERN` and performing the corresponding `ACTION` for the matches. If there is no `PATTERN`, then the `ACTION` will be performed on each line. But if there is no `ACTION` then the default `ACTION` (printing all lines) on the matching `PATTERN` will be performed (empty braces `{}` without any `ACTION` turns off default printing).

A simplest *eg*. would be

`awk '{print}' FILENAME` *try* `awk '{print}' AT_genes.gff`

Here, since there is no `PATTERN`, the print `ACTION` will be performed on each line (equivalent to `cat INFILE`).

Some inbuilt variables of `awk` include:

`FS`	Field Separator (default SPACE)
`OFS`	Output Field Separator (default SPACE)
`NF`	Number of Fields in the input
`NR`	Number of Records (lines) in the input
`RS`	Record Separator (default NEWLINE)
`ORS`	Output Record Separator (default NEWLINE)
`FNR`	File line number
`N`	Nth field of the line where N can be any number (eg. `$0` = entire line, `$1` = First field, so on)

`awk` accepts all standard patterns (regular expression and expression) plus some special patterns

`BEGIN`	Special PATTERN that is executed before the INPUT is read
`END`	Special PATTERN that is executed after the INPUT is read
`empty`	nonexistent PATTERN that matches every input record

Some simple *eg.* using `awk` (you can try these commands with AT_genes.gff FILE)

`awk NF FILE`	Deletes all blank lines
`awk 'NF > 0' FILE`	Deletes all blank lines
`awk 'NF > 4' FILE`	Prints only lines with more than 4 fields
`awk '$NF > 4' FILE`	Prints only lines with value of the 4th filed > 4
`awk 'END { print $NF }' FILE`	Prints value of the last field of the last line
`awk 'NR==25,NR==100' FILE`	Prints lines between 25 and 100
`awk 'NR==50' FILE`	Prints 50th line of input

`awk 'NR < 26' FILE`	Prints first 25 lines
`awk 'NR > 25' FILE`	Prints file after 25th line
`awk 'END { print NR }' FILE`	Prints the last line of the file
`awk '{ print NF ":" $0 }' FILE`	Prints number of fields in front of every line
`awk '{ print FNR ":" $0 }' FILE`	Prints line number in front of every line
`awk '$5 == "abc123"' FILE`	Prints lines which have 'abc123' in 5th field
`awk 'BEGIN { ORS="\n\n" }; 1' FILE`	Double spaces the file
`awk '{ print $1, $2 }' FILE`	Prints only 1st and 2nd field
`awk '{ print $2, $1 }' FILE`	Prints only 2nd and 1st field (swapping columns)
`awk '{ $2 = ""; print }' FILE`	Prints the file without 2nd column
`awk '/REGEX/' FILE`	Prints all the lines having REGEX
`awk '!/REGEX/' FILE`	Prints all the lines not having the REGEX
`awk '/AAA|BBB|CCC/' FILE`	Prints all the lines having either AAA, BBB or CCC
`awk 'length > 50' FILE`	Prints line having more than 50 characters
`awk '/POINTA/,/POINTB/' FILE`	Prints everything between POINTA and POINTB

Try to understand the following command lines (and record it):

awk 'END { print $NF }' AT_genes.gff	_____
awk 'NR==30,NR==35' AT_genes.gff.	_____
awk 'NR==25' AT_genes.gff	_____
awk 'NR<25' AT_genes.gff	_____
awk 'END { print NR }' AT_genes.gff	_____
awk '{ print NF ":" $0 }' AT_genes.gff > with_fields.txt	_____
awk '{ print NR ":" $0 }' AT_genes.gff > with_Line_num.txt	_____
awk '{ print $1, $3 }' AT_genes.gff	_____
awk '{print $1"\t"$3"\t"$2}' AT_genes.gff	_____
awk '{print $1,$2,$(NF-4),$(NF-3)} ' AT_genes.gff.	_____
awk '/Chr1/' AT_genes.gff.	_____

TR

The `tr` (*tr*anslate) utility in UNIX can translate or transliterate the input to produce a modified output. It uses 2 sets of parameters, and replaces occurrences of the characters in the first set with the corresponding elements from the other set.

```
tr [OPTIONS] "STRING1" "STRING2" <INFILE >OUTFILE
```

Useful options are

`-c`	complements the set of characters specified by string1
`-d`	delete occurrences of string1 (string2 not needed)
`-s`	squeeze repeats or multiple occurrences found in string1 will be replaced with one string2

Common uses of `tr` command are:

`tr "a-z" "A-Z"` or `tr "[:lower:]" "[:upper:]"`
Convert lower case to upper case (or upper to lower case)

`tr -s '\n'`
Convert each sequence of repeated newlines to a single newline

Files generated in both Mac and Windows OS will have a different `newline` character (to mark the end of the line) that is not recognized by the UNIX OS. Similarly files generated in UNIX will have a different newline that can't be read in Windows or Mac OS. The '`tr`' command provides an easy way to convert these 'newlines' to different forms.

`tr '\r' '\n' <MAC.TXT >UNIX.TXT`
Convert Mac text file to UNIX text file

`tr '\n' '\r' <UNIX.txt >MAC.TXT`
Convert UNIX text file to MAC text file

`tr -d '\015' <WIN.TXT >UNIX.TXT`
Convert Windows text file to UNIX text file

`tr '\n' '\015'<UNIX.txt >WIN.TXT`
Convert UNIX text file to windows text file

NOTE: There are in built commands like `dos2unix`, `mac2unix` and `unix2dos` to do these conversions automatically in most recent versions of `UNIX`.

There are several utilities that can mask low complexity regions of the genomes such as repeats. They do that either by converting the bases/residues to lower case (soft masking) or converting them to N or X (hard masking). The public databases often store these soft masked genomes. When downloaded it might be useful to remove the masking, if your analysis doesn't require it (pattern searching etc.). It can be easily done by changes cases

`tr "ATGC" "atgc" <AT_cDNA.fa >AT_cDNA_tr.fa`
Converts masking from the sequences and saves them in a new file

`tr "ATGC" "AUGC" <AT_cDNA.fa > AT_rna.fa`
Converts cDNA to mRNA sequence and saves them in a new file

WORD COUNT

`wc` (*w*ord *c*ount) is another useful command that lets you count the number of words (and lines) in a file

`wc FILENAME` *try this* `wc AT_genes.gff`
This outputs both number of words as well as lines in a file.

`wc -l FILENAME` *try this* `wc -l AT_genes.gff`
Outputs only number of lines in file

Often these commands are "piped" with other commands to count certain things. *eg.*: Counting the number of files in a directory, counting the number of sequences etc.

Task 2.6: Count how many files with .fa extensions are present in `sequences` directory.

```
ls Sequences| wc -l
```

SORT

`sort` command can be used to arrange things in a file. Simplest way to use this command is:

```
sort FILE1 > SORTED_FILE1
```

Useful options include

`-n`	numerical sort
`-r`	reverse sort
`-k N,N`	sort the Nth field (column), where N is a number. Sorting can also be done on the exact character on a particular field *eg.* `-k 4.3,4.4` sorts based on 3rd and 4th character of the 4th field. Additionally you can supply additional `-k` for resolving ties.
`-t`	specify the delimiters to be used to identify fields (default is TAB) *eg.* `-t` : to use ':' as delimiter

Task 2.7: The `Sequences` directory consists of numerically labeled files. UNIX can sort either alphabetically or numerically (not both) and hence they are arranged in `NT1.fa`, `NT10.fa`, `NT11.fa` *etc*. In order to sort them in an easy to read way, try using

```
ls |sort -n -k 1.3,1.4
```

This command lists all the files in sequences directory and then passes it to sort command. Sort command then sorts it numerically but only using 3rd and 4th letters of the first field (file name)

Try using sort on `AT_genes.gff` file

```
sort -r -k 1,1 AT_genes.gff
sort -r -k 4,4 AT_genes.gff
```

UNIQ

`uniq` (*uniq*ue) command removes duplicate lines from a sorted file, retaining only one instance of the running matching lines. Optionally, it can show only lines that appear exactly once, or lines that appear more than once. `uniq` requires sorted input since it compares only consecutive lines.

```
uniq [OPTIONS] INFILE OUTFILE
```

Useful options include

`-c`	count; prints lines by the number of occurrences
`-d`	only print duplicate lines
`-u`	only print unique lines

`-i` ignore differences in case when comparing
`-s N` skip comparing the first N characters (N=number)

Task 2.8: Number each lines based on number of occurrences:

```
uniq -c ids.txt
```

Print only duplicated lines.

```
uniq -d ids.txt
```

Print only unique lines.

```
uniq -u ids.txt
```

COMPARING FILES

`diff` (*diff*erence) reports differences between files. A simple example for `diff` usage would be

```
diff [OPTIONS] FILE1 FILE2
```

Useful options include

`-b` ignore blanks
`-w` ignore white space (spaces and tabs)
`-i` ignore case
`-r` recursively compare all files (when comparing folders)
`-s` list all similar files (when comparing folders)
`-y` side by side comparison of files

The differences reported will be in the form of corrections that are required to change the first file to second file

Generate `diffIDs.txt` by comparing the differences between `ids_a.txt` and `ids_b.txt`

```
diff -y ids_a.txt ids_b.txt > diffIDs.txt
```

Are these files different?

`comm` (*comm*on) command compares two sorted files line by line.

```
comm [OPTIONS] FILE1 FILE2
```

`-1` suppress lines unique to FILE1
`-2` suppress lines unique to FILE2
`-3` suppress lines that appear in both files

Task 2.9: Compare the same files (`ids_a.txt` and `ids_b.txt`) again with `'comm'` command and see how the outputs differ

```
comm -1 ids_a.txt ids_b.txt
```

```
comm -2 ids_a.txt ids_b.txt
comm -3 ids_a.txt ids_b.txt
```

DIVIDING FILES

`cut` divides the file into several parts and displays selected columns or fields from each line of a file. Normally cut command requires how the fields are separated and what fields needs to be displayed.

`cut -d "," -f 2-4 FILE` displays columns 2,3 and 4 of a file separated by ","

`cut -d "|" -f 1,10 FILE` displays 1st and 10th columns of a file separated by "|"

`cut -f 1 FILE` displays 1st column of a file, assumes TAB as delimiter

`split` generates output files of a fixed size (bytes or lines). Useful when huge file needs to be processed. *eg.,*

```
split -d -l 100 FILENAME SUFFIX
```

here `-d` specifies numeric suffix only (suffix00, sufix01, suffix02 *etc*.) while `-l` specifies number of lines in each file (100 in this case). If you want to split based on bytes, you can use `-b` option (*eg*. `-b 1k` or `-b 1m` for 1 KB and 1 MB respectively)

From the commands that you have learned, can you combine the all the split files into a single file again?

Task 2.10: Display only first column of the `AT_genes.gff` file using cut

```
cut -f 1 AT_genes.gff                (press ctrl + c to exit) or
cut -f 1 AT_genes.gff | less
```

Similarly, display 1st, 4th and 5th column of the `AT_genes.gff` file

```
cut -f 1,4,5 AT_genes.gff            (press ctrl + c to exit) or
cut -f 1,4,5 AT_genes.gff | less
```

Verify if all the columns in `AT_genes.gff` file has same number of entries in every field

```
cut -f 1 AT_genes.gff |wc -l
cut -f 2 AT_genes.gff |wc -l
cut -f 3 AT_genes.gff |wc -l
```

Split the file `AT_genes.gff` every 100,000 lines. Use `gff_split` as suffix for the files and use numerical suffix.

```
split -d -l 100000 AT_genes.gff gff_split
```

How many split files are generated: _____

```
ls gff_split* |wc -l
```

COMBINING FILES

`paste` prints lines consisting of sequentially corresponding lines of each specified file. eg.,

```
paste FILE1 FILE2 > FILE3
```

Combines the contents of `FILE1` and `FILE2`, side by side generating a new file, `FILE3`.

Task 2.11: Combine columns of `ids_a.txt` and `ids_b.txt` files.

```
paste ids_a.txt ids_b.txt
```

How many columns do you see after combining? _____

`join` combines two files based on the common field that is specified

```
join -t':' -1 N -2 N FILE1 FILE2
-t':'        Specify field separator (here ":" but you can specify anything. Default is TAB)
-1 N         Common field number (N) from the 1st file
-2 N         Common field number (N) from the 2nd file
```

Task 2.12: Join columns based on column 1 in `genes_a.gff` and column 3 in `genes_b.gff`

```
join -1 1 -2 3 genes_a.gff genes_b.gff
```

UNIX EXERCISE 3

This exercise mainly deals with using HPC clusters for large scale data (Next Generation Sequencing analysis, Genome annotation, evolutionary studies etc.). These clusters have several processors with large amounts of RAM (compared to typical desktop/laptop), which makes it ideal for running programs that are computationally intensive. The operating system of these clusters are primarily UNIX and are mainly operated via command line. All the commands that you have learned in the previous exercises can be used on HPC.

PREREQUISITES

ISU High Performance Computing (ISUHPC) offers shared cluster computing infrastructure for researchers and students at ISU. Brief descriptions for the available resources can be found here: http://www.hpc.iastate.edu/systems. To begin with, you need to request permission for accessing these resources either through your department or through your advisor. All workshop attendees will have their account setup on HPC class education cluster and they can use their ISU NetID and the password for logging-in. You should have already received a confirmation email about your account creation with instructions on how to connect to the cluster. In this exercise we will specifically teach you how to connect to a remote server (HPC), transfer files in and out of the server, and running programs by requesting resources.

LOGGING IN

You can log onto its front-end/job-submission system (`hpc-class.its.iastate.edu`) using your ISU NetID and password. Logging into HPC class requires an SSH client if you are using Windows but Mac/Linux have these built into their OS. There are several available for download for the Windows platform.

Microsoft Windows:

- **PuTTY** is an extremely small download of a free, full-featured SSH client.
- **SSH Secure Shell Client**, also a full featured client that is commercial. It is available as part of the Iowa State University site-licensed software.

Mac OS X/ Linux / Solaris or other 'nix systems

- The `ssh` command is pre-installed. You may start a local terminal window from "Applications > Utilities" or by searching for installed programs. Log in using

```
ssh -X username@hpc-class.its.iastate.edu
```

QUEUES

HPC class uses PBS for job scheduling and resource management. You will probably have access to the following queues, each with several nodes (1 node = 16 processors and 64 GB RAM).

```
short          1:00:00
medium         6:00:00
long_2node    73:00:00
large_short    0:15:00
tiny           0:10:00
long          72:00:00
```

FILE TRANSFER:

There are a number of ways to transfer data to and from HPC clusters. Which you should use depends on several factors, including the ease of use for you personally, connection speed and bandwidth, and the size and number of files which you intend to transfer. Most common options include `scp`, `rsync` (command line) and SCP and SFTP clients (GUI).

`scp` (*s*ecure *cop*y) is a simple way of transferring files between two machines that use the SSH (Secure SHell) protocol. You may use `scp` to connect to any system where you have SSH (login) access. `scp` is available as a protocol choice in some graphical file transfer programs and also as a command line program on most Linux, UNIX, and Mac OS X systems. `scp` can copy single files, but will also recursively copy directory contents if given a directory name. `scp` can be used as follows:

```
scp sourcefile username@hpc-class.its.iastate.edu:somedirectory/
```
(to a remote system from local)

```
scp username@hpc-class.its.iastate.edu:somedirectory/sourcefile
destinationfile
```
(from a remote system to local)

```
scp SourceDirectory/ username@hpc-class.its.iastate.edu:somedirectory/
```
(recursive directory copy to a remote system from local)

`rsync` is a fast and extraordinarily versatile file copying tool. It can synchronize file trees across local disks, directories or across a network

```
rsync -rave "ssh -l username" path/to/SourceDirectory username@hpc-
class.its.iastate.edu:somedirectory/
```
Synchronize a local directory with the remote server directory

```
rsync -rave "ssh -l username" username@hpc-
class.its.iastate.edu:SourceDirectory/ path/to/Destination/
```
Synchronize a remote directory with the local directory

User friendly (GUI) choices for file transfer:

- WinSCP (http://winscp.net): for Windows only
- FileZilla (https://filezilla-project.org): Windows/Linux/Mac
- **Cyberduck** (http://cyberduck.io): Mac and Windows
- Macfusion (http://macfusionapp.org): Mac only

VARIABLES

When your account is setup, some standard variables (known as environment variables) that are specific to your account were created. These variables can be used to simplify your navigation (many environment variables specify storage locations and paths). Think it of as "shortcut" that you create on your desktop to open the desired application that you frequently use. Your login automatically defines these variables for you. Some standard variables are

Name	Description
USER	your username
HOME	path to your home directory
PWD	path to your current directory
PATH	all directories searched for commands/applications
HOSTNAME	name of the machine you are on
SHELL	your current shell (bash, tcsh, csh, ksh)
SSH_CLIENT	your local client's IP address
TERM	type of terminal or terminal emulator being used

To perform the action you need to use them with `$` sign in front. For example:

```
cd $HOME
```
Changes your directory from the current location to `home` your directory

You can look up the values stored in these variables by using echo command
```
echo $VARIABLE_NAME
```

You can add any number of such variables manually by editing the hidden file (`.bashrc`) in your home directory (make sure that you create a backup copy of this original file before you start editing).

PRE INSTALLED PROGRAMS

To use pre-installed applications you can use the module command. First configure it using following command:

```
module use /shared/bioinformatics/modules
```

After that, you can use the `module load` command to load the software you want to use. For instance, to use FASTQC (program to check the quality of fastq reads of NGS) program,

```
module load fastqc
```

To check all available programs:

```
module avail
module what-is
```

SUBMITTING JOBS

To submit a job (running your script, starting a program etc) to the HPC-class cluster, you should use portable batch system (PBS) job scheduler. It will manage schedule jobs to run on HPC depending on the hardware requirement and other factors to efficiently use the available resources. If you run any jobs without the PBS then jobs will be executed on a front-end login host that is shared by all users. This will negatively impact everyone's ability to use HPC.

Usually, a submission script specifying the requirements of hardware for your job will be used to submit jobs on HPC. This script file is a simple text file where you specify:

- Memory requirement
- Desired number of processors
- Length of time you want to run the job
- Type of queue you want to use (optional)
- Additionally, you can also specify where to write output and error files as well as give name for your job while running on HPC

A simple job submission script is shown below:

```
#!/bin/bash
#PBS -l vmem=64Gb,pmem=8Gb,mem=64Gb
#PBS -l nodes=1:ppn=8:ib
#PBS -l walltime=48:00:00
#PBS -q long_2node
#PBS -o BATCH_OUTPUT -e BATCH_ERRORS
#PBS -N JOBNAME
```

You can also create a script using this html utility http://hpcgroup.public.iastate.edu/HPC/hpc-class/hpc-class_script_writer.html

It is useful to keep a 'template' of a job submission file in your home directory, which can be modified every time you submit a new job. Heavily customized template submission file with some useful features is given below:

```
#!/bin/bash
#PBS -q <queuename>
#PBS -l vmem=64Gb,pmem=4Gb,mem=64Gb
#PBS -l nodes=1:ppn=16:compute
#PBS -l walltime=48:00:00
#PBS -N <jobname>
#PBS -o ${PBS_JOBNAME}.o${PBS_JOBID} -e ${PBS_JOBNAME}.e${PBS_JOBID}
#PBS -m ae -M userid@iastate.edu
cd $PBS_O_WORKDIR
ulimit -s unlimited
echo "################ STATS ##################"
SSECS=$(date +"%s")
echo ${SSECS}
START=$(date +"%r, %m-%d-%Y")
echo -e "Host\t\t: $(hostname)"
echo -e "Processors\t: $(wc -l < $PBS_NODEFILE)"
echo -e "Nodes\t\t: $(uniq $PBS_NODEFILE | wc -l)"
echo -e "Total memory\t: $(free | grep Mem | awk '{print $2/1048576}' OFMT="%2.2f") Gb"
echo -e "Free memory\t: $(free | grep Mem | awk '{print $4/1048576}' OFMT="%2.2f") Gb"
echo -e "Directory\t: $(pwd)"
echo "#########################################"
module use /shared/bioinformatics/modules
module load moduleaname
## SCRIPT-START ##############################


## SCRIPT-END ################################
echo "############ TIME STAMP ################"
DIFF=$((`date +"%s"`-${SSECS}))
printf "Start\t\t:${START}\nEnd\t\t:$(date +"%r, %m-%d-%Y")\nTIME (hh:mm:ss)\t:%02d:%02d:%02d\n"
"$((${DIFF}/3600))" "$(((DIFF%3600)/60))" "$(((DIFF%3600)%60))"
echo "#########################################"
```

Whenever you submit a job, you have to modify: the numbers for memory/nodes/processors/walltime, program name and insert the script that you wish to run. Jobs can then be submitted using `qsub` command:

```
qsub template_jobfile.sub
```

A sample job to check the quality of the reads obtained from a sequencing project is present in the `jobfile.sub`. It is set to run it on `short` queue. To start the job:

```
qsub jobfile.sub
```

You will receive a confirmation `1234.hpc-class.its.iastate.edu` where `1234` is your job ID. Once you have submitted the job script, you can view status of jobs by using following commands:

`qstat –f yourjobid`	for information about your submitted job
`qstat -an1 yourqueuename`	current jobs running on queue you have submitted
`qstat –u yourusername`	list all the current jobs you are running on cluster
`qstat –a –u yourusername`	displays the status of your job

Additional resources:
http://hpcgroup.public.iastate.edu/HPC/hpc-class

Upon completion of the job, you will see many files in your working directory. Two of these files that start with your jobname are error log file (`jobname.e1234.hpc-class.its.iastate.edu`) and

output log file (`jobname.o1234.hpc-class.its.iastate.edu`). The fastqc results for two reads will be in two separate files (`R1_fastqc.html` and `R2_fastqc.html`). These folders are also saved as zip files by the program.

To view the results, just open `R1_fastqc.html` and `R2_fastqc.html` file. You can do this by

```
firefox R1_fastqc.html
```

DOWNLOADING DATA

In order to start using the computational power of the HPC cluster, you need to first get the data there. If your data is already in your local computer, you can transfer them easily using WinSCP software or any other software (refer prerequisites). But if the data that you will be using is available in the public databases then you can directly get it from there using `wget` command (*W*WW *get*)

To download data from NCBI Sequence Read Archive (SRA) or genomics core website or any other website:

```
wget http://website.url
```

As an example, we will download *Glycine max* (soy bean) annotation information file from Phytozome DB.

```
wget http://goo.gl/CDXx15
```

This is a single line command and you will see `‘Gmax_189_annotation_info.txt.gz’` file after few seconds. You can extract it and view it or delete it using the commands you have learnt.

QUICK REFERENCE SHEET

COMMANDS USED IN THIS MANUAL

Command	Function	Syntax/example usage
Navigation		
ls	list contents	ls [OPTIONS] DIRECTORY
pwd	print working directory	pwd
cd	change directory	cd ~ or cd #home directory
		cd .. #previous (parent directory)
File/Directory operations		
mkdir	make directory	mkdir DIRECTORY
cp	copy files/directories	cp SOURCE DESTINATION
man	manual page (help)	man COMMAND
mv	move files/directories	mv SOURCE DESTINATION
touch	create file	touch FILE
nano	edit file	nano FILE
less	view file (with more options)	less FILE
more	view file (with less options)	more FILE
cat	catalog file contents	cat FILE
head	show first few lines of a file	head FILE
tail	show last few lines of a file	tail FILE
rmdir	remove empty directory	rmdir DIRECTORY
rm	remove file(s)	rm FILE
Compression/archiving		
zip	zip compress	zip OUTFILE.zip INFILE.txt
		zip -r OUTDIR.zip DIRECTORY
unzip	decompress zipped file	unzip ANYTHING.zip
tar	archive and compress files/directories	tar -czvf OUTFILE.tar.gz DIRECTORY #compress
		tar -xzvf OUTFILE.tar.gz # extract
gzip	gzip files	gzip SOMEFILE
gunzip	decompress gzipped files	gunzip SOMEFILE.gz
File permissions		
chmod	change permissions for files/directories	chmod [OPTIONS] RELATIONS[+/-]PERMISSIONS FILE
File manipulations		
grep	search a pattern	grep [OPTIONS] "PATTERN" FILENAME
sed	stream edit a file	sed 's/search/replace/g' FILENAME
awk	multi-purpose command	awk 'PATTERN {ACTION}' FILENAME
tr	translate or transliterate a file	tr [OPTIONS] "STRING1" "STRING2" <INFILE
wc	word count	wc FILENAME
sort	sort files	sort FILE1 > SORTED_FILE1
uniq	display unique lines	uniq [OPTIONS] INFILE > OUTFILE
diff	display difference	diff [OPTIONS] FILE1 FILE2
comm	display common lines among files	comm [OPTIONS] FILE1 FILE2
cut	break files vertically based on fields	cut –d "DELIMITER" –f NUMBER FILE
split	break files horizontally	split [OPTIONS] FILENAME
paste	combine files side by side	paste FILE1 FILE2 > FILE3
join	join files based on common field	join -t'DELIMITER' -1 N -2 N FILE1 FILE2

ADDITIONAL COMMANDS

Command	*Function*
`du -sh DIR`	show directory size
`whoami`	display username
`date`	system date/time
`cal`	calendar
`find . -name FILE`	find a file/directory
`which CMD`	display default cmd path
`whereis CMD`	show possible locations of cmd
`locate FILE`	find instances of a file
`clear`	clear screen
`sleep 5`	pause 5 (any) seconds
`top`	current running processes
`ps`	current running processes
`wget URL`	download specified URL

NANO SHORTCUTS

Commands	*Function*
`ctrl+r`	read/insert file
`ctrl+o`	save file
`ctrl+x`	close file
`alt+a`	start selecting text
`ctrl+k`	cut selection
`ctrl+u`	uncut (paste) selection
`alt+/`	go to end of the file
`ctrl+a`	go to start of the line
`ctrl+e`	go to end of the line
`ctrl+c`	show line number
`ctrl+_`	go to line number
`ctrl+w`	find matching word
`alt+w`	find next match
`ctrl+\`	find and replace

PRE-DECLARED VARIABLES

*Variables**	*Description*
`$USER`	username
`$HOME`	home path
`$PWD`	working directory path
`$PATH`	path for executables
`$HOSTNAME`	machine name
`$SHELL`	current shell
`$SSH_CLIENT`	local client's IP address
`$TERM`	type of terminal

* env *command lists all the assigned variables*

SHORTCUTS

Commands	*Function*
TAB	autocomplete names
UP/DOWN	browse previous commands
`ctrl+c`	interrupt/kill anything
`ctrl+l`	clear screen
`ctrl+d`	quit, exit
`ctrl+z`	suspend (use `fg` to restore)
`!!`	repeat last command
`alt+.`	last argument of previous cmd
`ctrl+insert`	copy selection
`shift+insert`	paste copied text
`ctrl+a`	go to start of the line
`ctrl+e`	go to end of the line
`ctrl+r`	reverse search history
`cd ~`	go to home

PIPES, REDIRECTS

Redirects	*Function*
`cmd < file`	use file as input
`cmd > file`	write output to file
`cmd >> file`	append output to file
`cmd 2> stderr`	error output to file
`cmd 1>&2 file`	send output and error to file
`cmd1 | cmd2`	send output of cmd1 to cmd2

HPC-CLUSTER SPECIFIC COMMANDS

Command	*Function*	*Syntax/example usage*	
`qstat`	lists current jobs on queues	`qstat –an1 queue`	*# current jobs in specified queue*
		`qstat –u username`	*# current jobs by the user*
		`qstat –f jobid`	*# information about the job (id#)*
		`qstat -q`	*# list all queues*
		`qstat -a`	*# list all jobs*
		`qstat –r`	*# list running jobs*
		`qstat -Qf queue`	*# information about the 'queue'*
`qdel`	delete job from the queue	`qdel jobid`	
`qsub`	submit job to the queue	`qsub submissionfile.sub`	
`qhold`	hold job in queue	`qhold jobid`	
`qrls`	release job for running	`qrls jobid`	
`qnodes`	details about the nodes	`qnodes`	
`module`	use preinstalled programs	`module load PROGRAM`	*# loads program for use*
		`module list`	*# lists all loaded modules*
		`module avail`	*# lists available modules*
		`module unload PROGRAM`	*# unloads module*

PS: An A-Z Index of the Bash command line for Linux can be found at http://ss64.com/bash/index.html