

Perl for Biologists

Session 1

March 4, 2015

Introduction

Jaroslav Pillardy

Organization

- Perl for Biologists consists of 15 sessions, one every week, until June 10th
- Sessions will be taught by different Bioinformatics Facility staff members, the speakers are listed on the workshop web pages
- Slides will be posted online before each session.
- Please feel free to contact us with any questions:
 - Workshop coordinator: Jaroslaw Pillardy jp86@cornell.edu, Rhodes 623
 - Each session's speaker name is listed on session web page
 - You can find us in the Bioinformatics Facility directory
<http://cbsu.tc.cornell.edu/staff.aspx>
- You can carry out practical exercises on your own machine/laptop/desktop or use our BioHPC Lab workstations allocated for you. Machine allocations are posted online on workshop pages
<http://cbsu.tc.cornell.edu/ww/1/Default.aspx?wid=59>
- No programming experience necessary.

Organization

- BioHPC Lab machines are reserved for you and available all the time between now (March 4th) and June 21st (end of day June 20th)
- Please DO NOT use them for extensive calculations. It is fine to run on them any “light” Perl-related calculations, create and test Perl programs etc.
- You can see your reservations after logging into BioHPC Lab website <http://cbsu.tc.cornell.edu/>
- Helpful links:
 - Lab Users guide <http://cbsu.tc.cornell.edu/lab/use.aspx>
 - My reservations <http://cbsu.tc.cornell.edu/lab/labresman.aspx>
 - Reset password <http://cbsu.tc.cornell.edu/lab/labpassreset.aspx>
- Useful books:
 - “Learning Perl”, Randal Schwartz, Brain D Foy, Tom Phoenix
 - “Beginning Perl for Bioinformatics”, James Tisdall

Organization

“Perl for Biologists” office hours will be held

each Tuesday 11am-1pm and 3pm-4pm in 623 Rhodes.

Please don't hesitate to come if you have any questions or want to further discuss course topics.

Organization

- The workshop has practical examples and exercises.
- You can follow examples during the lecture, or you can carry them out afterwards.
- If you have any problems with them contact us or come to office hours
- **The only way to learn programming is to try!** Please do after lecture exercises – they are always discussed at the beginning of the next session.
- You can practice Perl programming on any computer, including your Windows or Mac laptop.
- We will focus on our Linux machines since it is most likely environment on which you will run your future Perl programs.
- Therefore next few slides are “Linux primer”.

Connecting to Linux b machines

Text-based connection: ssh (Secure SHell)

GUI (graphical) connection: X-Windows or VNC

Please refer to the following document for more information about GUI connections

[http://cbsu.tc.cornell.edu/lab/doc/Introduction to BioHPC Lab v2.pdf](http://cbsu.tc.cornell.edu/lab/doc/Introduction%20to%20BioHPC%20Lab%20v2.pdf)

Logging in to a Linux machine

- ❑ On any Linux machine, you need
 - **network name** of the machine (e.g. cbsumm10.tc.cornell.edu)
 - an account, i.e., **user ID** and **password**
 - on your local computer: remote access software (typically: **ssh client**)

- ❑ Linux is a multiple-access system: multiple users may be logged in and operate on one machine at the same time

Logging in to a Linux machine

❑ Remotely from a PC via **ssh** client

- Install and configure remote access software (**PuTTY**).
- Use **PuTTY** to open a terminal window on the reserved workstation using **ssh** protocol;
- You may open several terminal windows, if needed.

Logging in to a Linux machine

❑ Remotely from other Linux machine or Mac via native **ssh client**

➤ Launch the Mac's terminal window. Type

```
ssh jarekp@cbsuwrkstX.tc.cornell.edu
```

(replace the “**cbsuwrkstX**” with the workstation that you just reserved, and “**jarekp**” with your own user ID). Enter the lab password when prompted.

➤ You may open several terminal windows, if needed, and log in to the workstation from each of them.

Logging in to CBSU machines from outside of Cornell

Two ways to connect from outside:

- ❑ Install and run the CIT-recommended the VPN software (<http://www.it.cornell.edu/services/vpn>) to join the Cornell network, then proceed as usual
- ❑ Log in to `cbsulogin.tc.cornell.edu` (or `cbsulogin2.tc.cornell.edu`):

`ssh jarekp@cbsulogin.tc.cornell.edu` (using PuTTY or other ssh client program)

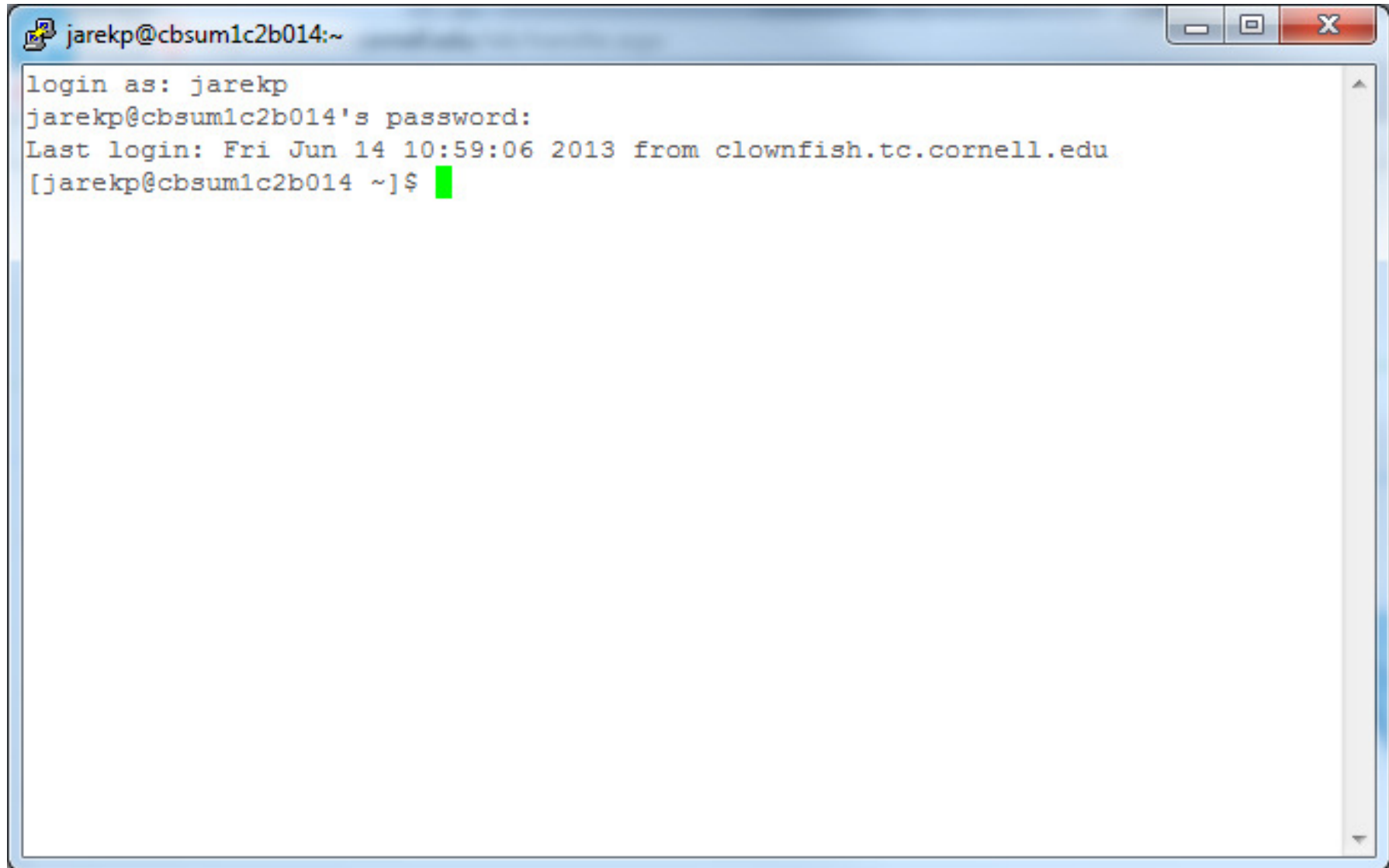
Once logged in to `cbsulogin`, ssh further to your reserved machine

`ssh jarekp@cbsuwrkst3.tc.cornell.edu`

Backup login machine is `cbsulogin2.tc.cornell.edu`

<https://cbsu.tc.cornell.edu/lab/doc/BioHPCLabexternal.pdf>

Terminal window

A terminal window with a blue title bar. The title bar contains a small icon on the left and three window control buttons (minimize, maximize, close) on the right. The main area of the window is white and contains text from a login process. The text is as follows: 'login as: jarekp', 'jarekp@cbsum1c2b014's password:', 'Last login: Fri Jun 14 10:59:06 2013 from clownfish.tc.cornell.edu', and '[jarekp@cbsum1c2b014 ~]\$' followed by a green cursor block. A vertical scrollbar is visible on the right side of the terminal area.

```
jarekp@cbsum1c2b014:~  
login as: jarekp  
jarekp@cbsum1c2b014's password:  
Last login: Fri Jun 14 10:59:06 2013 from clownfish.tc.cornell.edu  
[jarekp@cbsum1c2b014 ~]$
```

Terminal window

- ❑ User communicates with the machine via **commands** typed in the terminal window
 - Commands are interpreted by a program referred to as **shell** – an interface between Linux and the user. We will be using the shell called **bash** (another popular shell is **tcsh**).
 - Typically, each command is typed in one line and “**entered**” by hitting the **Enter** key on the keyboard.
 - Commands deal with **files** and **processes**, e.g.,
 - request information (e.g., list user’s files)
 - launch a simple task (e.g., rename a file)
 - start an application (e.g., Firefox web browser, BWA aligner, IGV viewer, ...)
 - stop an application

Logging out of a Linux machine

- ❑ While in terminal window, type **exit** or **Ctrl-D** - this will close the current terminal window

How to access BioHPC Lab machines

BioHPC Lab User's Guide

<http://cbsu.tc.cornell.edu/lab/userguide.aspx>

Slides from workshop “Introduction to BioHPC Lab”

[http://cbsu.tc.cornell.edu/lab/doc/Introduction to BioHPC Lab v2.pdf](http://cbsu.tc.cornell.edu/lab/doc/Introduction%20to%20BioHPC%20Lab%20v2.pdf)

Slides from workshop “Linux for Biologists”

[http://cbsu.tc.cornell.edu/lab/doc/Linux workshop Part1.pdf](http://cbsu.tc.cornell.edu/lab/doc/Linux%20workshop%20Part1.pdf)

[http://cbsu.tc.cornell.edu/lab/doc/Linux workshop Part2.pdf](http://cbsu.tc.cornell.edu/lab/doc/Linux%20workshop%20Part2.pdf)

Programming languages

- Strongly typed vs. Loosely typed (context based)
all variables declared
C, C++, Java, C# Perl, Python, Visual Basic
- Scripted (interpreted) vs. Compiled
Executed “on the fly”, by line
Perl, Visual Basic, Shell Python, Java, C# C, C++, Fortran
- Flat vs. Object oriented
No complex objects
C, Pascal Perl, Java, C#, C++

Programming languages

Perl is a loosely typed, interpreted, object-oriented programming language .

Loosely typed:

Easier to write, more flexible, no need for extra code to “cast” variables. VERY EASY to make errors. Perl variables are typed dynamically based on context.

Interpreted:

More portable – will execute anywhere where interpreter is present IF program does not require specific libraries and IF it doesn't use system specific commands. MUCH slower, automatic code optimization impossible.

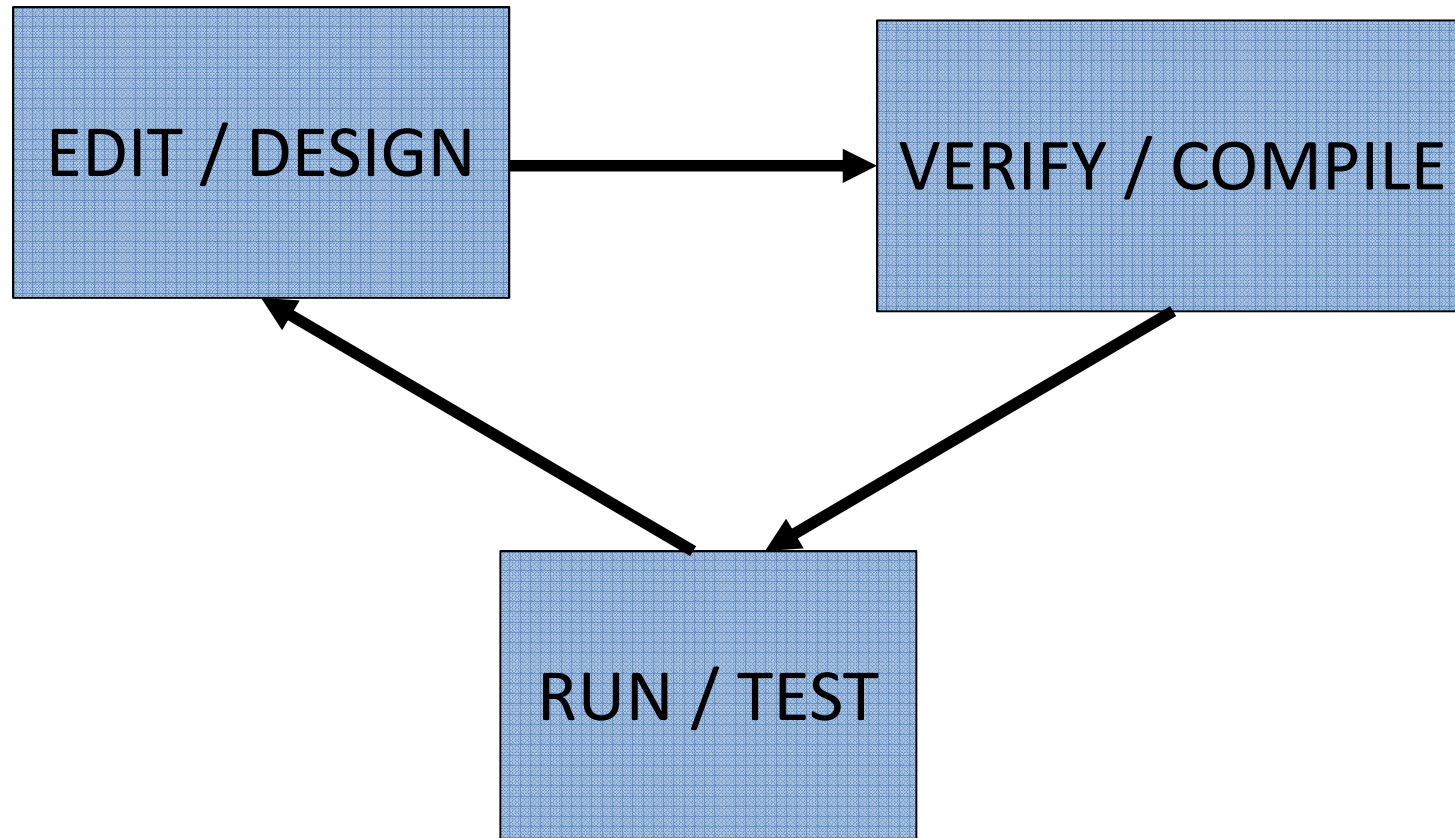
Object-oriented:

Program can be compartmentalized with reusable code. Very powerful way to solve problems. Slower.

Why Perl?

- Easy to learn, fast to write (rapid prototyping), informal
- High-level – compact code, lots of useful functions
- Huge public library of code available that can be directly used
- Runs anywhere (with some caution)
- Flexible: useful for scripting, websites as well as large programs
- Perl is not fast, but excellent to “stich” together other programs – very good for pipelines, task automation, interacting with OS.
- Perl can be easily used to perform various “in-between” functions like process control, file/data control and conversion, string operations, database operations and many more

Programming cycle



Perl programs are scripts – text files interpreted line by line

Need to use TEXT editor to create and edit them

TEXT file is a file than uses only letters, numbers and common symbols plus “new line” or “tab” special characters. NO formatting or other binary code (MS Word vs. text example).

Plain ASCII characters: byte codes between 32 and 126
(byte => 8 bits, 0-255; 1 bit => smallest unit of information)

Modern text files can use special characters (e.g. ó or ö) and symbols (e.g. ß or §) with Unicode – and Perl can work with them too. But they MUST be used with a TEXT editor (and better yet – not used at all 😊)

Example: Notepad and Word

ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com

ASCII Table

128	Ç	144	É	160	á	176	☐	192	Ł	208	⌌	224	α	240	≡
129	ü	145	æ	161	í	177	☐	193	Ł	209	⌌	225	β	241	±
130	é	146	Æ	162	ó	178	☐	194	Ł	210	⌌	226	Γ	242	≥
131	â	147	ô	163	ú	179		195	Ł	211	⌌	227	π	243	≤
132	ä	148	ö	164	ñ	180	┆	196	—	212	┆	228	Σ	244	∫
133	à	149	ò	165	Ñ	181	┆	197	┆	213	┆	229	σ	245	∫
134	â	150	û	166	²	182	┆	198	┆	214	┆	230	μ	246	÷
135	ç	151	ù	167	°	183	π	199	┆	215	┆	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	┆	200	⌌	216	┆	232	Φ	248	°
137	ë	153	Ö	169	┆	185	┆	201	┆	217	┆	233	⊗	249	.
138	è	154	Ü	170	┆	186		202	⌌	218	┆	234	Ω	250	.
139	ì	155	◊	171	½	187	┆	203	┆	219	■	235	δ	251	√
140	í	156	ℓ	172	¼	188	┆	204	┆	220	■	236	∞	252	∞
141	î	157	⌘	173	¡	189	┆	205	=	221	┆	237	φ	253	²
142	Ä	158	⌘	174	«	190	┆	206	┆	222	┆	238	ε	254	■
143	Å	159	ƒ	175	»	191	┆	207	┆	223	■	239	∧	255	

Source: www.LookupTables.com

TEXT Editors

vi

- Available on all UNIX-like systems (Linux included), i.e., also on lab workstations (type **vi** or **vi file_name**)
- Free Windows implementation available (once you learn vi, you can just use one editor everywhere)
- Runs locally on Linux machine (no network transfers)
- User interface rather peculiar (no nice buttons to click, need to remember quite a few keyboard commands instead)
- Some love it, some hate it

gvim

- Vi (see above) with a graphical interface – X-Windows needed. Windows version available.

nano

- Available on most Linux machines (our workstations included; type **nano** or **nano file_name**)
- Intuitive user interface. Keyboard commands-driven, but help always displayed on bottom bar (unlike in vi).
- Runs locally on Linux machine (no network transfers during editing)

TEXT Editors

gedit (installed on lab workstations; just type **gedit** or **gedit file_name** to invoke)

- X-windows application – need to have X-ming running on client PC.
- May be slow on slow networks...

edit+ (<http://www.editplus.com/>)

- **Commercial product**
- Runs on a local machine (laptop) and transfers data to/from Linux workstation as needed
- Can browse Linux directories in a Windows-like file explorer
- May be slow on slow networks
- Some people swear by it

emacs (installed on lab workstations)

Xcode (Mac)

Notepad (Windows)

TEXT Files on Unix, Windows and Mac

End-of-line problem:

• Unix:	\n	CR	10	0x0a
• Windows	\n\r	CR+LF	10 13	0x0a 0x0d
• Mac (old)	\r	LF	13	0x0d
• Mac (new)	\n	CR	10	0x0a

Make sure files transferred from one system to another are properly converted

On Linux there is a set of nice utilities

`unix2dos file_name`

`dos2unix file_name`

`unix2mac file_name`

`mac2unix file_name`

Example: Windows and Unix files on Windows

Vi basics

Opening a file:

vi my_reads.fastq (open the file my_reads.fastq in the current directory for editing; if the file does not exist, it will be created)

Command mode: typing will issue commands to the editor (rather than change text itself)

Edit mode: typing will enter/change text in the document

<Esc> exit edit mode and enter command mode (this is the most important key – use it whenever you are lost)

The following commands will **take you to edit mode**:

i	enter insert mode
r	single replace
R	multiple replace
a	move one character right and enter insert mode
o	start a new line under current line
O	start a new line above the current line

The following commands **operate in command mode (hit <Esc> before using them)**

x	delete one character at cursor position
dd	delete the current line
G	go to end of file
1G	go to beginning of file
154G	go to line 154
\$	go to end of line
1	go to beginning of line
:q!	exit without saving
:w	save (but not exit)
:wq!	save and exit

Arrow keys: move cursor around (in both modes)

Look of a typical Perl script:

```
#!/usr/local/bin/perl  
  
#this is my first Perl script  
  
print "Hello, CBSU\n";
```

“shebang” notation – path to the program to interpret the script, must be the first line and start with #!

`#!/usr/local/bin/perl`

`#this is my first Perl script`

`print "Hello, CBSU\n";`

statement ends with a
semicolon

function to print out text

anything starting with # is
a comment, unless it is #!
in the first line

“shebang” notation – path to the program to interpret the script, must be the first line and start with #!

parentheses can be always omitted, unless it changes the meaning of expression

```
#!/usr/local/bin/perl
```

```
#this is my first Perl script
```

```
print("Hello, CBSU\n");
```

statement ends with semicolon

anything starting with # is a comment, unless it is #! in the first line

function to print out text

Strings in Perl

- Sequence of characters – simple (ASCII) or extended (Unicode, wide)
- Special characters like NL or CR are represented as `\xxxx` (C notation)
 - `\n` new line (NL)
 - `\t` tab character
 - `\r` return (CR)
 - `\x0a` any character represented by hex number (0a = 10 = NL)
 - `\"` double quotation
 - `\'` single quotation
 - `\\` backslash

- Strings may be joined by `'.` operator

`"string 1 " . "string 2" <=> "string 1 string 2"`

- Some characters have special meaning in Perl, most prominently `$` and `@`
 - `\$` {dollar}
 - `\@` {at}

Strings in Perl

- Single Quoted

Single quoted strings have LITERAL meaning – no special characters are recognized:

<code>'string 1'</code>	<code>string 1</code>
<code>'string 1\n'</code>	<code>string 1{backslash}n</code>
<code>'\string 1\''</code>	<code>'string 1'</code>
<code>' string 1\\1 '</code>	<code>string 1\1</code>

- Double-Quoted

Double quoted strings do interpret special characters properly:

<code>"string 1\n"</code>	<code>string 1{new line}</code>
<code>"\string 1\""</code>	<code>"string 1"</code>

Perl installation and usage depends on the OS

External Perl libraries (modules) are accessible via CPAN

CPAN = **C**omprehensive **P**earl **A**rchive **N**etwork

You can download and use any of publicly available modules in
your programs

Perl on Linux

- Almost always installed as a part of the system, if not ask your system admin
- Usually it is `/usr/bin/perl` or `/usr/local/bin/perl`
- May be several versions installed, each with its own libraries and features
- Version can be checked with command
`>perl -v`
`>/usr/bin/perl -v`
- If you need a particular Perl installation in your program, write it into the first line
`#!/usr/local/special/bin/perl`
- If you need **default** Perl installation in your program, write it into the first line
`#!/usr/bin/env perl`
- Once invoked, Perl interpreter knows where its system-wide modules reside

Perl on Linux

Execute Perl program

- If the scripts has executable right

>./script_name.pl

>./script_name.pl >& output

- Regardless of executable right

>perl script_name

- Compile (verify) Perl program

>perl -c script_name

Make script executable:

>chmod u+x script_name

Perl on Linux

If you need custom modules located in a custom place:

- write it into first line

```
#!/usr/local/bin/perl -I /home/jarekp/my_modules
```

- set environmental variable

```
PERL5LIB=/home/jarekp/my_modules:/usr/another/path/lib; export PERL5LIB
```

- Execute explicitly with Perl interpreter and options

```
>perl -I /home/jarekp/my_modules my_script.pl
```

Lets write and execute the script NOW

```
#!/usr/local/bin/perl  
  
#this is my first Perl script  
  
print "Hello, CBSU\n";
```

Perl on Linux: CPAN

Two interfaces to CPAN

>cpan

>perl -MCPAN -e shell

Then you can type command

install modname	- install module modname
r modname	- report if upgrade is available
upgrade modname	- upgrade
m modname	- info about modname

Remember: there is a cpan for EACH Perl installation, make sure you are using right one

Perl on Linux: CPAN

If you want to install a module for your own use, without being an admin:

Configure cpan (only first time)

```
>cpan
o conf makepl_arg INSTALL_BASE=~/.myPERL_LIB
o conf mbuidl_arg INSTALL_BASE=~/.myPERL_LIB
o conf prefs_dir ~/.myPERL_LIB/prefs
o conf commit
```

Install module(s)

```
>cpan
install modname
```

Need to reset CPAN:

```
o conf init
```

Set up environment so Perl knows where to look

```
PERL5LIB=/home/jarekp/myPERL_LIB/lib/perl5:$PERL5LIB
Export PERL5LIB
```

Perl on Linux: CPAN

Local configuration example

Configure cpan (only first time)

```
>cpan
o conf makepl_arg INSTALL_BASE=/home/jarekp/perl5
o conf mbuild_arg INSTALL_BASE=/home/jarekp/perl5
o conf prefs_dir /home/jarekp/perl5/prefs
o conf commit
```

Set up environment so Perl knows where to look: edit /home/jarekp/.bashrc and add the following

```
export PERL_LOCAL_LIB_ROOT="$PERL_LOCAL_LIB_ROOT:/home/jarekp/perl5";
export PERL_MB_OPT="--install_base /home/jarekp/perl5";
export PERL_MM_OPT="INSTALL_BASE=/home/jarekp/perl5";
export PERL5LIB="/home/jarekp/perl5/lib/perl5:$PERL5LIB";
export PATH="/home/jarekp/perl5/bin:$PATH";
```

Perl on Windows

Recommended Perl is ActivePerl: <http://www.activestate.com/activeperl>

Download binary and install – choose free version.

“shebang” line of any script is ignored on Windows

Windows recognizes Perl scripts by extension .pl

There is a nice GUI to CPAN

Example of script and GUI

Perl on Mac

Similarly as on Linux it comes preinstalled on OS X.

All Linux information should apply.

A bit more complicated script

```
#!/usr/local/bin/perl

use warnings;
use Bio::Perl;

#this is my first Perl script

print "Hello, CBSU\n";
```

use ModuleName;

Declares usage of Perl module “ModuleName”, includes all proper definitions

use warnings;

Declares use of “warnings” module – Perl will now report any place it thinks is ambiguous or suspicious: same as >perl -w

use Bio::Perl;

Declares use of BioPerl module – more details later

“use” statement can be declared as a parameter of Perl interpreter

```
>perl -MBio::Perl
```

... and then something can be executed ...

```
>perl -MBio::Perl -e "print \"OK\\n\";"
```

If Bio::Perl is installed it will print "OK", otherwise an error will occur.

Easy way to check if a module is installed.

Example: CPAN installation of Template::HTML

Exercises

1. Write a Perl program that prints your name and e-mail in the following format in one line:
first_name last_name <emailaddr@domain.edu>
2. Are the following modules installed on your BioHPC Lab machine?

Net::Ping
XML::Special
Net::Telnet
CBSU::HDF5