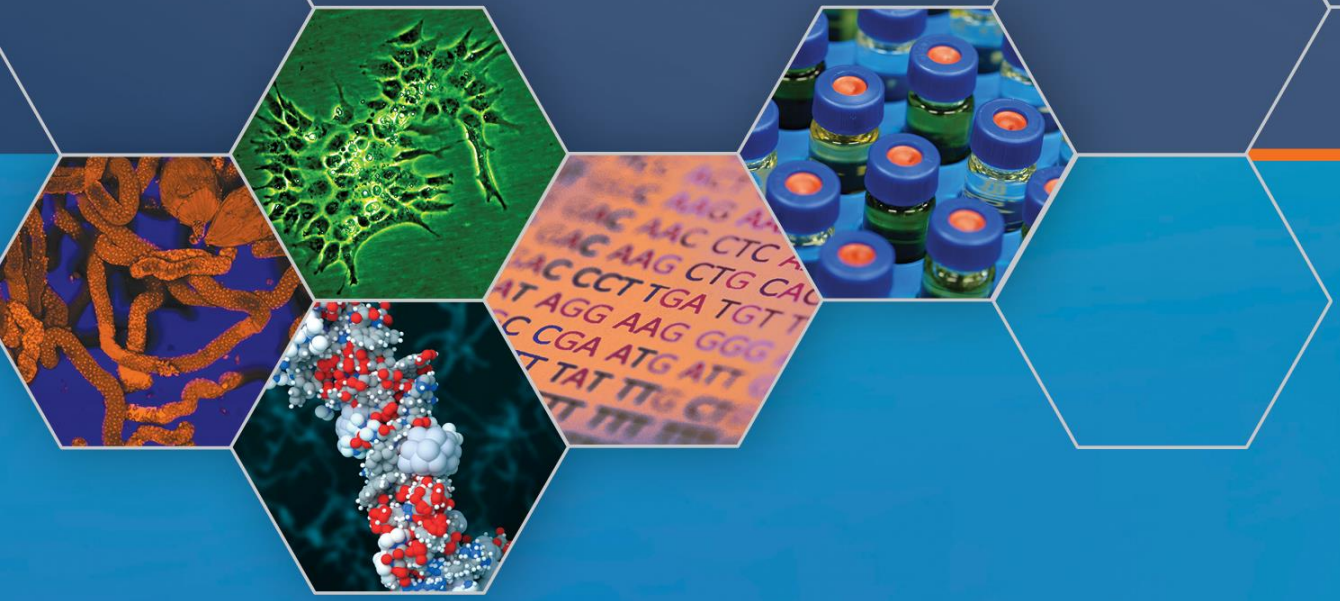
ICBR

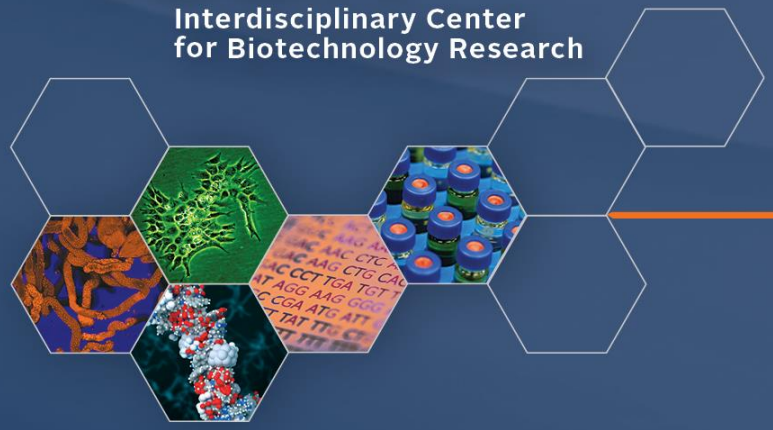
Interdisciplinary Center
for Biotechnology Research



UF | UNIVERSITY of FLORIDA

ICBR

Interdisciplinary Center
for Biotechnology Research



Bioinformatics 101 – Lecture 2

Introduction to command line

Alberto Riva (ariva@ufl.edu), J. Lucas Boatwright (jlboat@ufl.edu)

ICBR Bioinformatics Core

UF UNIVERSITY of FLORIDA

Computing environments

- Standalone application – for local, interactive use;
- Command-line – local or remote, interactive use;
- Cluster oriented: remote, not interactive, highly parallelizable.

Command-line basics

- Commands are typed at a *prompt*. The program that reads your commands and executes them is the *shell*.
- Interaction style originated in the 70s, with the first visual terminals (connections were *slow*...).
- A command consists of a program name followed by *options* and/or *arguments*.
- Syntax may be obscure and inconsistent (but efficient!).

Command-line basics

- Example: to view the names of files in the current directory, use the "ls" command (short for "list")

`ls`	plain list
`ls -l`	long format (size, permissions, etc)
`ls -l -t`	sort newest to oldest
`ls -l -t -r`	reverse sort (oldest to newest)
`ls -lrt`	options can be combined (in this case)

- Command names and options are **case sensitive**!

File System

- Unix systems are centered on the *file system*. Huge tree of directories and subdirectories containing all files.
- *Everything* is a file. Unix provides *a lot* of commands to operate on files.
- File extensions are not necessary, and are not recognized by the system (but may still be useful).
- Please do *not* put spaces in filenames!

Permissions

- Different privileges and permissions apply to different areas of the filesystem.
- Every file has an *owner* and a *group*. A user may belong to more than one group.
- Permissions specify read, write, and execute privileges for the owner, the group, everyone else. Example:
- `ls -l qmonitor`

```
-rwxr-x---  1 ariva riva 68 Sep 15  2015 qmonitor
```

Filesystem commands

- `pwd` where are we?
- `cd` move around the filesystem
- `ls` view contents of directory
- `mkdir` create new directory
- `chmod` change file permissions
- `cp` copy files
- `mv` move files
- `rm` delete files (use with caution!)
- `rmdir` remove (empty) directory

Every user has a *home* directory. “cd” with no arguments takes you back there.

Working with text files

- Plain text files are the “lingua franca” of Unix systems.
- Most files are structured, either in a simple way (rows and columns, comma- or tab-delimited) or complex (e.g. XML, JSON).
- Text files can be created manually with an editor (e.g. nano, vi) or generated by other programs.

Working with text files

Viewing files:

- `file` show type of file
- `cat` display contents of a file
- `more` display contents better
- `less` display contents even better
- `head` display top (10) lines
- `tail` display last (10) lines

Working with text files

Finding information on or in files:

- `wc` count rows / words / chars in file
- `grep` print rows containing text / pattern
- `cut` select columns from delimited file
- `diff` display differences between two files
- `find` search for files matching tests (file name, size, modification date, etc).

Working with text files

Other file operations…

- `sort` sort file lines (alphabetically, numerically, based on one or more fields)
- `uniq` detect (and remove / print / count) duplicate lines
- `split` split a file into chunks
- `paste` join files side by side
- `shuf` shuffle file lines

More advanced processing can be accomplished with `sed`, `awk`.

Other useful commands

- `echo` display message
- `screen` run multiple terminals over one connection
- `watch` execute a command periodically
- `man` get help on a command
- `gzip` compress files
- `gunzip` uncompress files
- `zcat/zmore/zgrep` view / search compressed files
- `zip` create zip archive
- `time` measure command execution time
- `wget` download file given URL (http, ftp, etc)

Scripting

The real power of unix commands lies in *scripting*:

- Sequences of commands can be saved to a file and repeated easily.
- Shell syntax supports variables and control structures (conditionals, loops, etc). Simple but powerful.
- Basic Unix philosophy: simple building blocks can be easily combined to create new complex tools.

Pipelining and redirection

- Pipelining: the output of a program can be directed to another program with the | (pipe) operator. Complex sequences of commands can be created easily.

- Redirection:

< read command input from a file

> send command output to a file

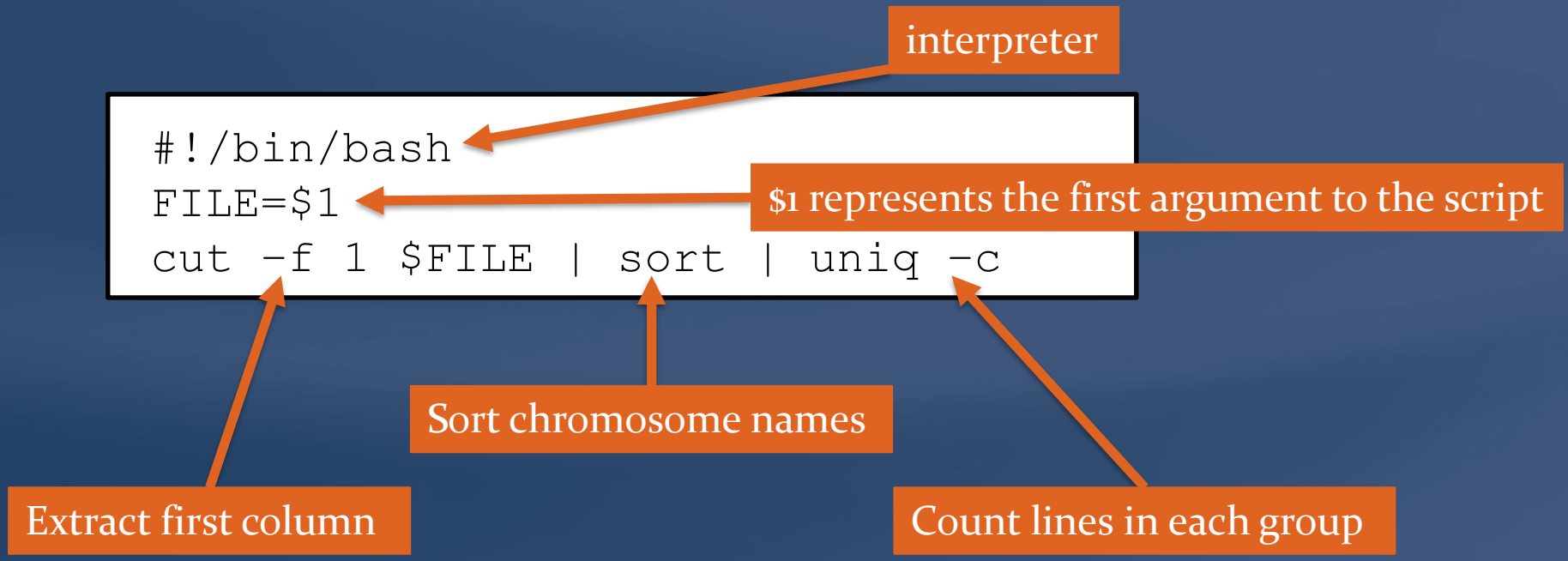
>> append command output to a file

- General form:

```
command1 < file1 | command2 > file3
```

Scripting example

- Given a tab-delimited file with chromosome names in the first column, the following script prints the number of occurrences of each chromosome:



How to create a script

- Use a text editor (e.g. nano, vim) to create the file. Use any name, extension not necessary (but .sh is a good choice).
- Make the script executable:
  `chmod +x scriptname`
- Put script in a directory that is in your PATH, or call it with full pathname.

Cluster-oriented approach

- Extension of shell scripting: the same commands are executed *in parallel* on a large number of nodes, with different arguments.
- The *scheduler* manages job submission and execution according to priorities, policies, available resources, and current usage.
- HiPerGator is a cluster computer with approx. 50,000 cores and several PB of memory. It uses *slurm* as the job scheduler.

Modules

- Software on HiPerGator is organized into *modules*. Module commands:

`module load <name>`	Load named module
`module spider <patt>`	Show matching module(s)
`module unload <name>`	Unload named module
`module purge`	Unload all modules

The **dibig_tools** module provides access to several useful commands. For example: `kut`, a more powerful version of `cut`.