(PRACTICAL)
COMPUTATIONAL PHYSICS

Physics 551
Lecture 1

CONTACT INFORMATION

- **Hugh Dickinson** (hughd@iastate.edu).
- **Office** A409 Zaffarano Hall.
- **Website**: Blackboard Learn - Physics 551
- **Textbooks**: None, but online resources may be recommended.

NOTATION

Extra Reading

Reading Exercise

Optional Exercise

- The lecture slides for this course will attempt to use a uniform notation throughout. A normal paragraph looks like this.

☞ *Italicized paragraphs with pen bullets will indicate definitions, with the defined word or phrase shown in* ***SMALL-CAPS****.*

✏ Pencil bullets will indicate the introduction of **new notation**.

☞ Pointing hand bullets indicate important points that might otherwise be overlooked.

ONGOING ASSESSMENT

- **Submission:**
  - Assessments will be submitted using using a combination of **VirtualBox** and **Git** (see later slides).
- **Homeworks:**
  - Weekly reading assignments, online quizzes, mini-projects involving techniques covered that week.
- **Midterm:**
  - Longer, **predefined** project using a combination of all the techniques covered so far.
  - Marks will be awarded for **functionality** (does it work?!), **code clarity** and **documentation**.

FINAL ASSESSMENT

- **Self-defined** project addressing a **real-world** problem using **all or most** of the techniques covered by the course. Production of a **user manual** for the software you develop.

- Marks will be awarded for **design** (does your project use appropriate tools and techniques for the task?) **functionality**, **code clarity** and **documentation**.

WHAT TO EXPECT

- A **practical** course about **general** tools and techniques used for scientific computation.
- In-class **demonstrations** and **case studies**.
- **Basic** introductions to the **Python**, **C++** and **NI LabView** programming languages.
- Introductions to **auxiliary tools** and **utilities** that are essential for effective programming.
- Instructions for writing **robust**, **reusable** and **comprehensible** code in a **collaborative** environment.

WHAT **NOT** TO EXPECT

- A **theoretical** course presenting e.g. a computer-scientific approach to algorithmic design.
- A course on **High Performance Computing** using parallel computing architectures.
- **Detailed** instruction on programming using any **specific** language.
- **Comprehensive** coverage of multiple programming **paradigms** - e.g. functional programming, C++ template meta-programming.
- Advanced code **profiling** and **optimization**.

DEMONSTRATION

Introducing VirtualBox

DESCRIBING THE FILE SYSTEM

☞ *The locations of files and directories within the **Linux** filesystem are defined using **PATHS***

✏ Paths consist of tokens separated by '**/**' characters. Paths will appear in monospaced, italicized type e.g.

`path/to/the/file`

☞ The rightmost token is the can be a file name **or** a directory name. Other tokens name directories.

DESCRIBING THE FILE SYSTEM

- Paths can be **relative** or **absolute** e.g.

☞ ***ABSOLUTE PATHS*** *are specified with a leading '/' character and describe the location of the file relative to the file system* ***ROOT****.*

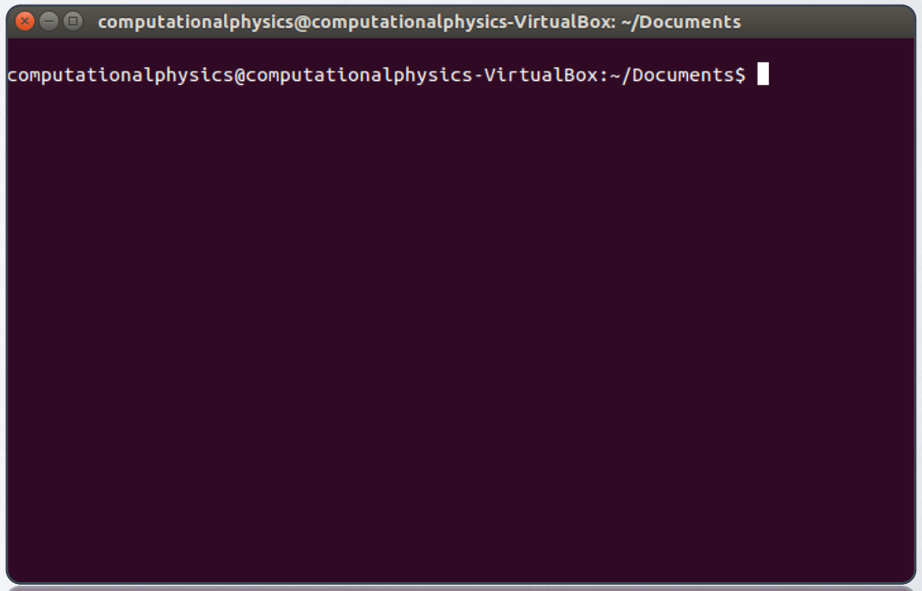
☞ ***RELATIVE PATHS*** *are specified without a leading '/' character and describe the location of the file relative to the* ***current directory****.*

TERMINALS AND THE SHELL

- Before the arrival of GUIs, interaction with computers was **text-based**.

- This **simple interface** remains **very useful** for scientific computation.

☞ *For historical reasons, utility programs that provide this type of interface are called* ***TERMINALS****.*



TERMINALS AND THE SHELL

- *The **SHELL** is the program that interprets and reacts to any textual commands that are entered into a terminal utility.*
- *Shell commands consist of **one or more** whitespace-separated **TOKENS**.*
- Shell commands will appear in `monospaced` type, on lines beginning with a `$` to indicate the shell prompt e.g.

```
$ shell_command
```

- Shell commands are **case-sensitive**.

SHELL COMMAND FLAGS

☞ *The action of some shell commands can be modified using **FLAG** (or **SWITCH**) tokens that are prepended by one **or** two hyphens e.g.*

```
$ shell_command -f --flag
```

- **Some** flags provide extra flexibility using value tokens that immediately follow them e.g.

```
$ shell_command -f v1 --flag=value2
```

☞ **Note** that flags with a double-hyphen prefix are **normally** separated from their **value** by an = character, **not whitespace**.

SHELL COMMAND ARGUMENTS

☞ *Many shell commands act on **inputs** that are specified using **ARGUMENT** tokens e.g.*

```
$ shell_command argument
```

- Argument and flag tokens can coexist within **the same** shell command e.g.

```
$ shell_command -f v1 –flag argument
```

☞ **Generally**, arguments and flags **always** follow the **shell command** but can **often** be listed in any order.

SHELL COMMANDS FOR FILESYSTEM INTERACTION

- **C**hange **d**irectory

```
$ cd
```

- **L**i**s**t directory contents

```
$ ls
```

- **C**o**p**y file (note arguments)

```
$ cp original_file copied_file
```

SHELL COMMANDS FOR FILESYSTEM INTERACTION

- **M**o**v**e (or rename) file **or** directory

```
$ mv origin destination
```

- **R**e**m**ove a **file** (**Warning!** No undelete)

```
$ rm doomedFile
```

- Display **man**ual for a shell command

```
$ man command
```

SHELL COMMANDS FOR INSPECTING TEXT FILES

- Dump contents of **one or more** files to the screen

```
$ cat text_files…
```

- Display a file line-by-line or page-by-page

```
$ more text_file
```

☞ Display the next **line** by pressing **ENTER**.

☞ Display the next **page** by pressing **SPACE**.

SHELL COMMANDS FOR APPENDING TO TEXT FILES

- **Set** or **reset** the contents of a text file

```
$ echo "New contents" > file
```

- **Append** a line to a text file

```
$ echo "Added line" >> file
```

☞ ***PIPING*** *passes the output of one shell command to be used as the input for another e.g.*

```
$ ls -l long_directory | more
```

DEMONSTRATION

Shell Basics

VERSION CONTROL

☞ *Fundamentally, a **VERSION CONTROL SYSTEM** (**VCS**) maintains a **record** of the **evolving** state of a filesystem.*

- Use of a VCS within your code development workflow enables:
  - Ongoing, incremental **backup** of your code as you develop it.
  - Straightforward **collaboration** with other developers.
  - Quasi-automatic **documentation** of the changes you or your collaborators make.
  - Straightforward **reversion** of broken code to a previously working state.

INTRODUCING GIT

- Several VCS implementations exist. Popular examples include CVS, Subversion and Mercurial.
- This course will make extensive use of the **Git** VCS.

☞ **You** will use **Git** to **maintain** the code you develop and to **retrieve** and **submit** homework assignments.

- Some homework assignments will require you to **collaborate** on joint projects.
- **Git** also helps **me** to monitor your progress and spot common problems.

WANT TO KNOW MORE?

- There are no official textbooks for this course. However…
- Git provides **extensive** documentation in the form of the **Git Pro** book, which is available online

http://git-scm.com/book/en/v2

- This course will make extensive use of material from the Git Pro book.
- The book is also available for **download** in **HTML**, **PDF**, **EPUB** and **MOBI** formats.

GIT BASICS

Git Pro Book
Chapter 1.3

☞ *A directory containing files that are being monitored by Git is called a* ***WORKING DIRECTORY****.*

- Every working directory contains a **hidden subdirectory** called **`.git`**.

☞ *A* ***HIDDEN DIRECTORY*** *or* ***HIDDEN FILE*** *has a name that begins with '**.**'. It will not appear in GUI windows.*

☞ Hidden files **can** be listed using the **`ls`** shell command with a special **`-a`** flag

```
$ ls -a
```

GIT BASICS

Git Pro Book
Chapter 1.3

- The ***.git*** directory contains a **database of snapshots** representing previous states of the working directory.
- It also contains **metadata** describing the change history of each file in the working directory.
- Simply altering files the working directory will **not** alter the contents of the ***.git*** directory.

☞ Rather, you **must** execute **several** shell commands to update the metadata.

GIT BASICS

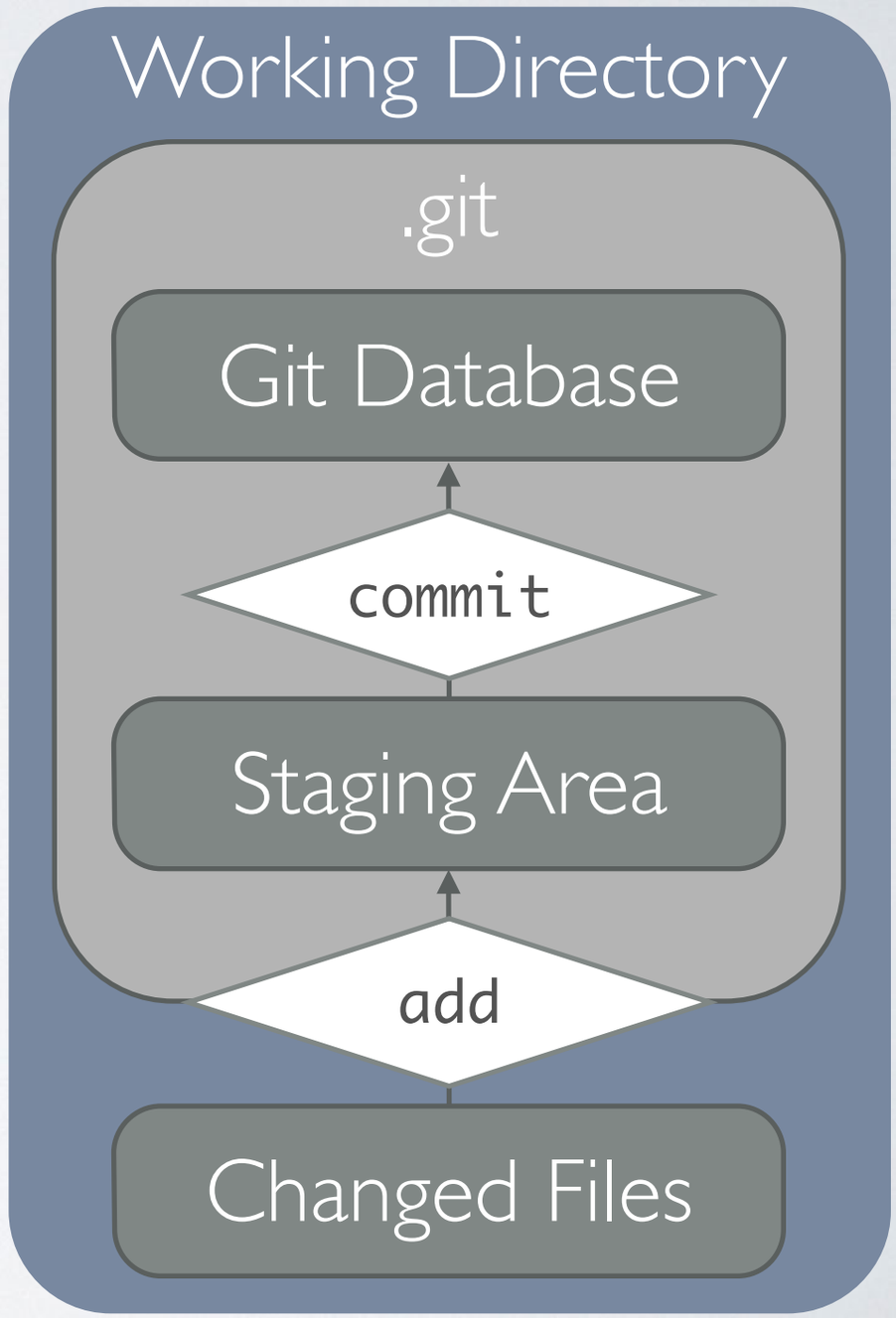
Git Pro Book
Chapter 1.3

- *The process of updating the* **`.git`** *directory is called* ***COMMITTING*** *changes.*

- Committing changes is a **two-stage** process that uses two shell commands

- *Altered files must be copied to a* ***STAGING AREA*** *using*

```
$ git add changed_files…
```



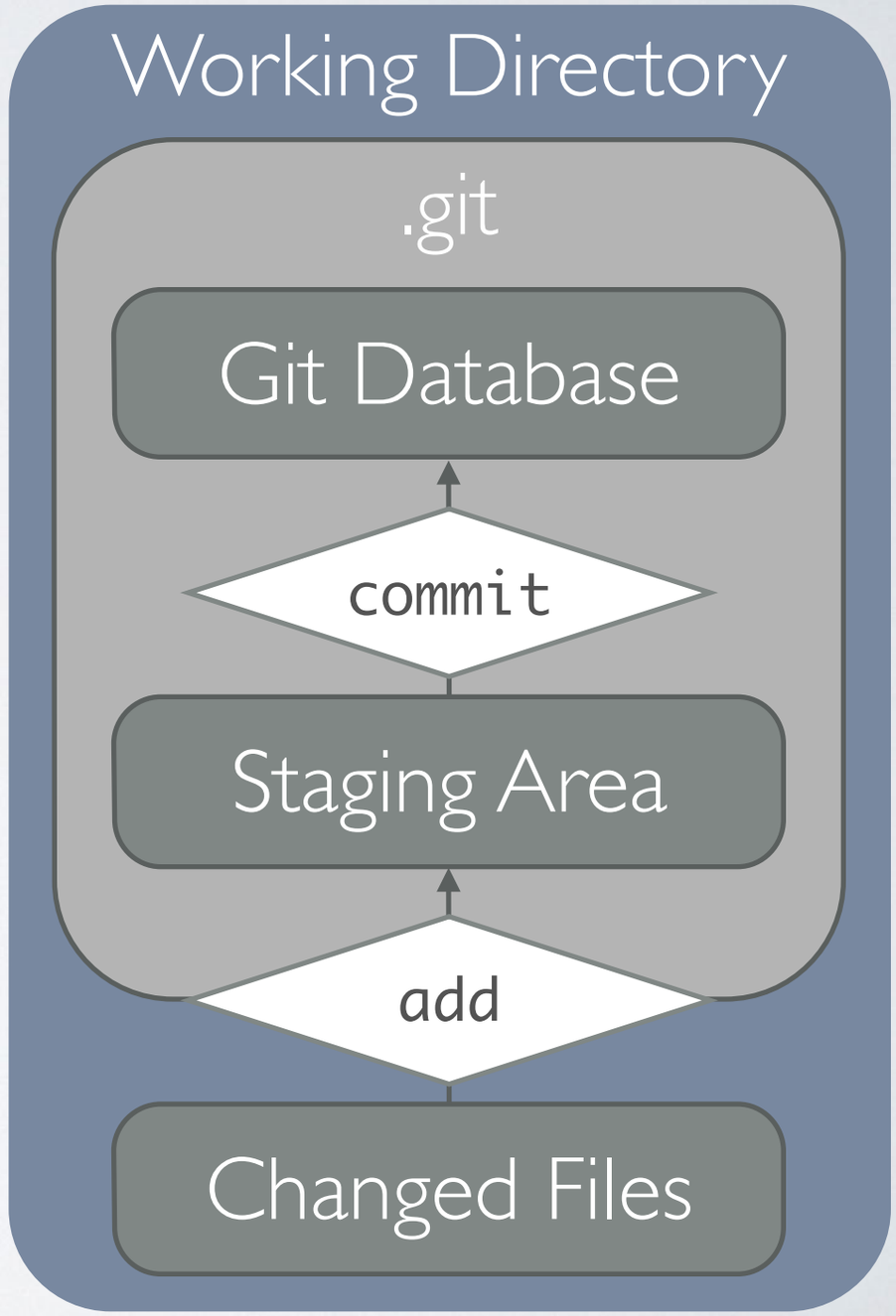
GIT BASICS

Git Pro Book
Chapter 1.3

- The staging area is used to assemble files before committing.
- Files in the staging area can be **permanently** recorded to the Git database using

```
$ git commit –m“message”
```

- The value of the `–m` flag should be a **short message** describing what has changed since the last commit.



DEMONSTRATION

Basic Git - Committing changes

LECTURE 1 SUMMARY

- After reviewing the material in this lecture you should know:

  1. How to launch the **Ubuntu Linux** virtual machine **ComputationalPhysics** using **VirtualBox**.
  2. How to open a **Terminal** once the virtual machine is running.
  3. What is meant by the terms **shell**, **shell command**, **flag** (or switch) and **argument**.

LECTURE 1 SUMMARY

4. How to **view the manual** for a particular shell command.
5. How to **navigate the Linux filesystem** using shell commands.
6. How to **list the contents of directories** using shell commands.
7. How to **copy**, **move, inspect** and **append to** files using shell commands.

LECTURE 1 SUMMARY

8. How to view **hidden files** and **directories**.
9. A basic definition of **version control**.
10. How to **add** files in a Git working directory to the **staging area**.
11. How to **commit** staged files to the Git snapshot database.

LECTURE 1 HOMEWORK

Read Chapters 1.1, 1.2 and 1.3 from the Git Pro Book

Watch the "What is Version Control" video at http://git-scm.com/video

- Your first homework mini-project is preloaded onto your Ubuntu Linux virtual machine.
- It is designed to give you some practice interacting with the shell.
- Begin by launching a terminal and navigating to:

  `Documents/theShellGym/lecture1/homework`

- You can find further instructions by inspecting the `instructions` file which is contained within the `instructions` subdirectory.
- Good Luck!