Welcome to Imperial

Remote Working for Computing Students

Nuri C. (nuric@imperial.ac.uk)

Slides available at https://www.doc.ic.ac.uk/~nuric

Contents

- The terminal
- Lab machines
- Secure Shell (SSH)
- Home folders
- Secure Copy (SCP)
- SSH Keys
- Editing files
- Multiple terminals (tmux)

Don't worry if none of these ring a bell, they shouldn't. At least they didn't for me. This material will just give basic pointers for you so you can hopefully explore later.

Screencasts

I would recommend watching the screencasts as you read but if you would like to jump to the screencasts, here are the links:

- The Terminal basics and how to run commands / programs.
- SSH & Home Folder for connecting to the college and navigating around your college home folder.
- Secure Copy for transfering files securely over SSH.
- SSH Keys to authenticate securely without our password.
- Editing Files for coding courseworks.
- Multiple terminals to work more efficiencly over a single connection.

Terminal

A *computer terminal* is a device, hardware that is used for interacting with a computer. Mainly thought of as a monitor and keyboard. In the past, these were teletypes: you would type something, and the computer (after some significant time and noise) would reply back. National Computing Museum and Centre for Computing History has working terminals, including some old game consoles from my childhood..

A **terminal emulator** is a computer program that emulates the terminal often giving a text based input and output. They run another program called the **shell** which is responsible for reading your commands, making sense + running them and giving you the result.

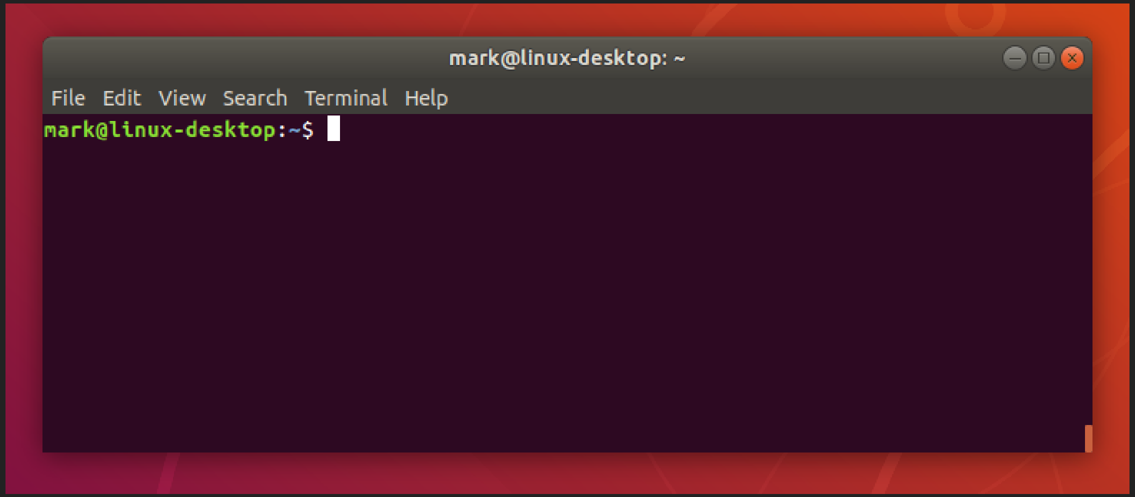
```
SLMIN .SYS    12P 20-Dec-85    VM    .SYS     3P 13-Aug-86
XL    .SYS     4P 20-Dec-85    LD    .SYS     8P 23-Aug-86
SP    .SYS     6P 13-Aug-86    DL    .SYS     5P 13-Aug-86
RT11SJ.SYS    78P 13-Aug-86    DU    .SYS     8P 13-Aug-86
NL    .SYS     2P 13-Aug-86    TT    .SYS     2P 13-Aug-86
SQ    .SYS     5P 31-May-85    RL02DG.SYS    71P 21-Nov-84
BASIC .SAV    56  24-May-79    BINCOM.SAV    24  20-Dec-85
DATIME.SAV     4  20-Dec-85    DIR   .SAV    19  20-Dec-85
DUMP  .SAV     9  20-Dec-85    DUP   .SAV    47  20-Dec-85
TSXMOD.SAV    78  27-Nov-92    FORTRA.SAV   206  21-May-85
HARRIS.SAV    41  12-Jun-85    LET   .SAV     5  20-Dec-85
START .P36     2  21-Dec-91    RETRO .OBJ  1519P 16-May-88
EM1A  .STM    19P 02-Feb-93    TSXUCL.TSX    22  07-Mar-92
STAND .LIN    12P 15-Aug-83    TSXV6 .MSG     1P 04-Sep-95
EM1B  .STM    19  11-Feb-93    JKFLIP.SAV    30  08-Mar-96
JKFLIP.FOR     3  08-Mar-96    RT11FB.SYS    93P 20-Dec-85
ANNOT .SAV    38  18-Apr-93    TSXP23.NEW  1200P 27-Nov-92
DUO   .DIR    18  16-Jul-96    DL    .DIR     7  16-Jul-96
DIR   .DIF    26  16-Jul-96    DEMOFG.OBJ     1   -BAD-
DEMOBG.OBJ     1   -BAD-       EVAN  .ID      1
 114 Files, 5949 Blocks
 14433 Free blocks

.o
```

DEC VT100 computer terminal, RIP.

Before graphical user interfaces (GUI), interaction with a computer was through text only. It was very concise because of limitations on screen size, how much we can read and type. All operating systems have a terminal emulator with a shell program:

- **Ubuntu** (Linux family operating systems, UNIX-like): Comes with GNOME Terminal emulator and runs GNU Bash as its default shell
- **macOS** (Apple MacBooks, UNIX-like): Comes with its own terminal emulator running Z shell (zsh)
- **Windows** (?): Has many attempts, latest being PowerShell

The idea is the same: input commands -> computer runs them -> gives output



```
mark@linux-desktop:~$
```

The GNOME Terminal in Ubuntu from linux command line tutorial.

We can do pretty much everything in the terminal. But why should we when we can look at nice user interfaces? One example is browsing the internet, we *can* browse the internet with a text based browser but we prefer not to. The terminal is:

- **concise**: you spend less time clicking, moving the mouse, actually you can get rid of the mouse. The commands are shortened and less verbose.
- **fast**: you do not wait for the terminal / shell, it waits for you to do something.
- **practical**: as we will see, just working in text mostly allow us to easily work remotely and safely.

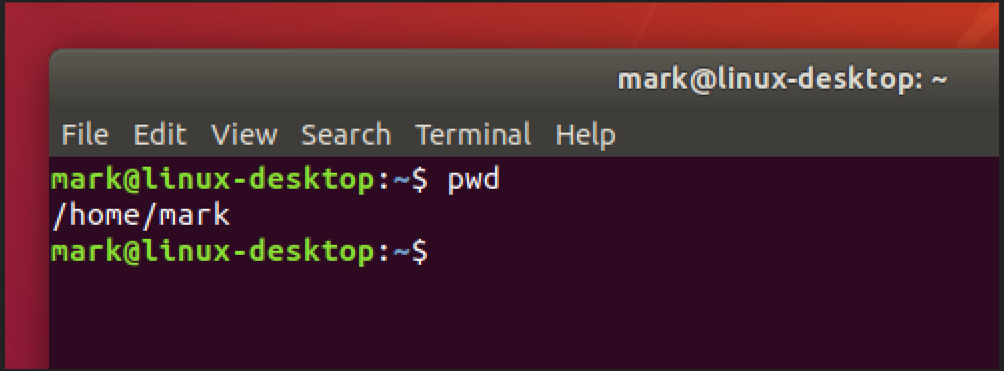
We will cover Linux terminal commands, some of these also work on Windows PowerShell, but we will focus on Linux since the computing labs in the college run Ubuntu operating system.

Get a terminal going

- **Ubuntu**: you can search for the terminal app or launch one using Ctrl+Alt+T keyboard shortcut.
- **macOS**: similarly search for the terminal app and run it.
- **Windows**: to get a feel for a terminal you can run PowerShell, or the Linux environment such as Windows Linux Subsystem which pretends to be Ubuntu, but inside Windows.

Don't worry about having a Linux terminal if you can't setup one, after a brief listing of commands we will focus on connecting to the college computing lab machines which all run Ubuntu.

When you open one, you should be presented with a **prompt** that is ready for you to type some commands. You can customise this prompt, but often we include things like the current user, machine name and directory you are in.



```
mark@linux-desktop:~$ pwd
/home/mark
mark@linux-desktop:~$
```

Here, the prompt is `mark@linux-desktop:~$` and the command we typed is `pwd`, we will see what `pwd` is shortly.

Right, let's get to it with some of the most used commands. Here I use folder and directory interchangeably. I curated these based on what I use almost everyday:

- `ls [DIRECTORY]`: list folder contents, what is here? If you don't specify the folder it will list current directory. A shell program can only be in a single folder at a time.
- `pwd`: print working directory, where is here?
- `cd [DIRECTORY]`: change directory to specified directory or to your home folder if you don't specify.
    - There are some special directories, "." refers to the current directory, ".." refers to parent directory (one up) and "~" is the your home folder.
- `echo [STRING]`: echoes what you tell it, "echo Hello World" will print "Hello World". Seems useless but very useful.

- `cat FILE`: dumps the content of files, you can also use `less`
- `rm [FILE]`: remove the file, use `rm -r` for removing a directory *recursively*.
- `mv SOURCE DEST`, `cp SOURCE DEST`: move or copy files from source to destination, you can specify `-r` to move or copy folders.
- `man COMMAND`: show the manual for a program, try running `man man` :)
- `exit`: quits the shell program and the terminal. Yes, this one is quite useful.

When a program is stuck, you can type `Ctrl+C` to *interrupt* it. Try stopping the command `yes hahaha` using `Ctrl+C` repeatedly.

Strongly recommend taking your time and following a tutorial.

> Instead of memorising commands and how to use them, think of what *you* want to do and then search online for how it might be done.

Some more commands and programs that I often use:

- `mkdir [DIRECTORY]`: create a new directory
- `touch [FILE]`: create an empty file
- `ssh`: secure shell, we will see this later!
- `find`: find files or folders with patterns, for example `find . -name "foo*.txt"` which will find all text files starting with "foo" in the current folder.
- `grep PATTERN FILE`: find a pattern inside a file, for example `grep foo atextfile.txt` will search for occurrences of "foo" inside the text file.
- `git`: is a version control program, it allows us to track changes in files and code. You will use it with coursework etc.

We can *redirect* input and output of these programs to and from files using the redirection operator `>`. For example, `ls > myfiles.txt` will write the output of the `ls` command into the file. We can read it with `cat myfiles.txt`, note that the `>` operator *overwrites* the file, you also instead use the append operator `>>` with will append to the file. You can read more online.

Another very useful thing is to *pipe* the input and output of the commands, effectively chaining them. For example, `sort myfiles.txt | uniq | wc -l` will sort, *then* find the unique ones, *then* count the number of lines. You can learn about piping and commands such as `uniq` online or using `man uniq`.

Here is a quick way of writing something to a file:

```
echo "This is the first line" > myfile.txt
```

Video Walkthrough: The Terminal

LINK

Lab Machines

We have a large computing lab in the department on the ground floor with hundreds of desktop workstations. There is a quiet section, tutorials section and a brand new room opened this year. The lab machines provide a connected environment within the department, anyone can login to any machine and work, your profile and files *roam* over the network file shares.

Note that we run a separate infrastructure to the rest of the college which uses Windows. Only DoC members and students taking courses can access the lab machines and department resources. More info is at the department website.

A typical day in labs.

Computing Support Group

These resources, lab machines, storage and accounts etc. are managed by the wonderful Computing Support Group (CSG). I encourage you to visit their webpage for guides, list of services they offer and status updates.

Some information about lab machines:

- They have enough resources to carry out academic work: Intel i7, 16GB RAM etc.
- They run latest stable long-term support version of Ubuntu.
- They are for academic use only.
- There are 200+ of them in the computing lab.
- We do not have root / admin access, you cannot just install packages or run commands as root.
- They are shared across all students, you can use anyone of them and be mindful of others.

Secure Shell (SSH)

Our aim is now to remotely connect to one of those lab machines. We will achieve this using Secure Shell (SSH) protocol.

> SSH gives us a remote terminal on the machine we are connecting to.

https://www.imperial.ac.uk/computing/people/csg/guides/remote-access/ssh/

SSH is a protocol and there are many programs that implement it. The most common is the OpenSSH set of programs which include a server (that the lab machines run) and a client program `ssh` which is available on Ubuntu, macOS and Windows (enabled as extra features).

For Windows, you can also use PuTTY which is a common SSH client. It comes with a host of programs to handle various aspects of SSH.

Chromebooks have an SSH App too. So does Android and iOS. If you can get comfortable working through a terminal then you can almost work from anywhere.

For Windows 10, Ubuntu and macOS users `ssh` should be pre-installed on the terminal.

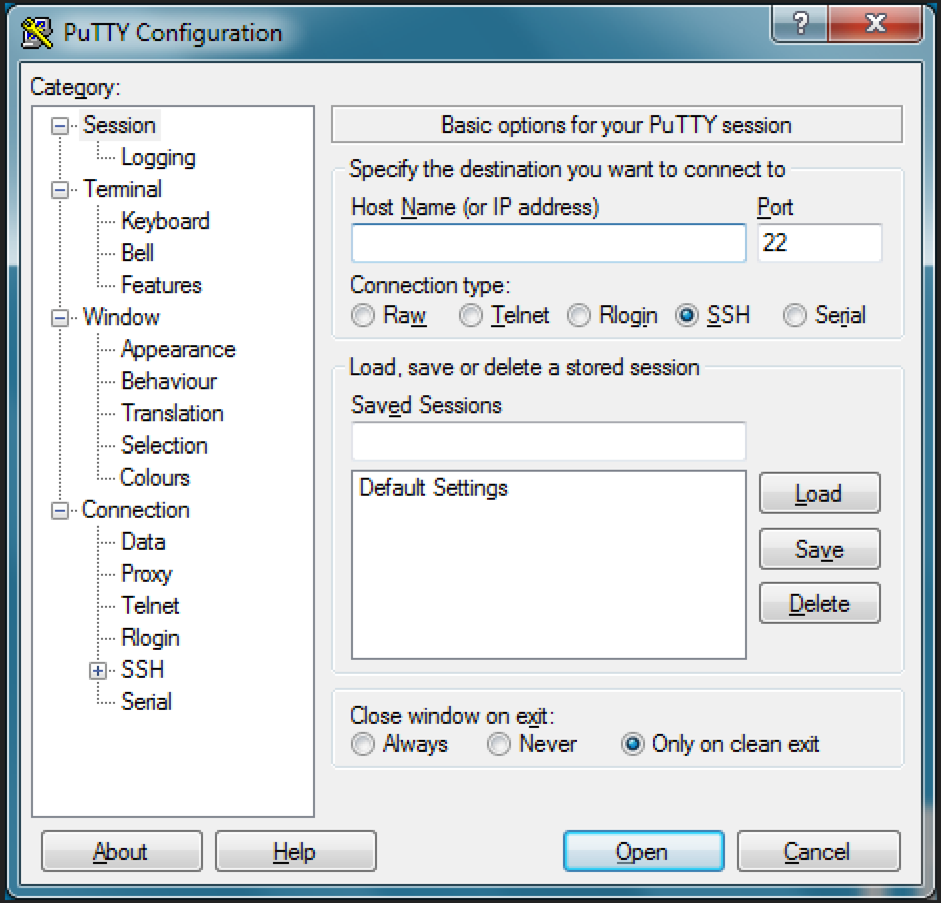
If you would like to use PuTTY and related programs, please install it from https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html using the "Windows Installer" and follow the instructions.

On Ubuntu, macOS and Windows 10, using SSH is as simple as running:

```
ssh [USERNAME@]shell1.doc.ic.ac.uk
```

- `ssh` is the command itself, the program we are going to run
- `[USERNAME@]` is your college username, if it is the same as the local machine you can omit it. I'm using [] to mean optional.
- `shell1.doc.ic.ac.uk` is the machine / hostname we are trying to connect to.

After running for the first time, it will ask us if we trust this machine, we do. Then ask for our college password. And that's it, we're in..



For clients like PuTTY, you will see an interface that asks for the same information. Most importantly the hostname, PuTTY then asks for the username and password.

Shell vs Lab Machines

None of the lab machines are exposed to the internet, in other words you cannot directly access them. You need to be inside the college network from an entry point: department shell servers. This is not a technical term, just what we call the set of machines that are exposed to the internet:

- shell1.doc.ic.ac.uk
- shell2.doc.ic.ac.uk
- ...
- shell5.doc.ic.ac.uk

Pick one at random. They are all the same and provide an entry point into the department network. From these machines we will *jump* or connect to an actual physical lab machine. **Do not run anything intensive on these shell servers**, they are for basic file and entry point jobs.

SSH again to a lab machine

Now that we are inside the department network, we can connect to an actual lab machine to work on. We have a basic program that automatically selects a free lab machine and connects for you:

- `sshtolab`: will find an empty machine and just ssh again into that one for you.

this script could be run from shell servers or lab machines. **If the command is not found, try `/vol/linux/bin/sshtolab`**.

A shortcut for macOS, Ubuntu and Windows 10 users

You can directly ask ssh to run this command on connection and connect to a lab machine. On your own machine:

```
ssh -t [USERNAME@]shell3.doc.ic.ac.uk /vol/linux/bin/sshtolab
```

and you will first connect to a shell server, then redirected to a free lab machine.

A little help for trusting machines

By default, when you try to connect to a new lab machine, it will ask if you trust it. We can say we trust all `*.doc.ic.ac.uk` machines by adding it to our `ssh` configuration. *After connecting to the college you can run the following two commands:*

```
mkdir -p ~/.ssh
echo "Host *.doc.ic.ac.uk
    StrictHostKeyChecking no" > ~/.ssh/config
```

which says do not check hosts (machines) that end with the department domain.

Home Folders

After connecting to a shell server or a lab machine, if you print the current directory using `pwd`, you'll see something like:

```
/homes/[your user name]
```

This is your *home folder*, your personal storage space to work within. Whichever machine you login to, shell or lab machine, you will have your home folder there, any changes will be synced.

Some notes about your home folder:

- It has a quota, 8GB and 40k files maximum. Can check using `quota -Q -s`
- It is a network file share that is loaded whenever you login to machine.
- It is *backed up daily*, so try to clean up regularly. For example, zip old courseworks and upload to OneDrive.
- It is assumed to be private, you change file / folder permissions yourself to let others read it, i.e. Linux file permissions.

Video Walkthrough: SSH & Home Folder

LINK

Secure Copy (SCP)

A common scenario when working remotely is we would like to transfer files to and from our local machine to our home folder. There are again two options:

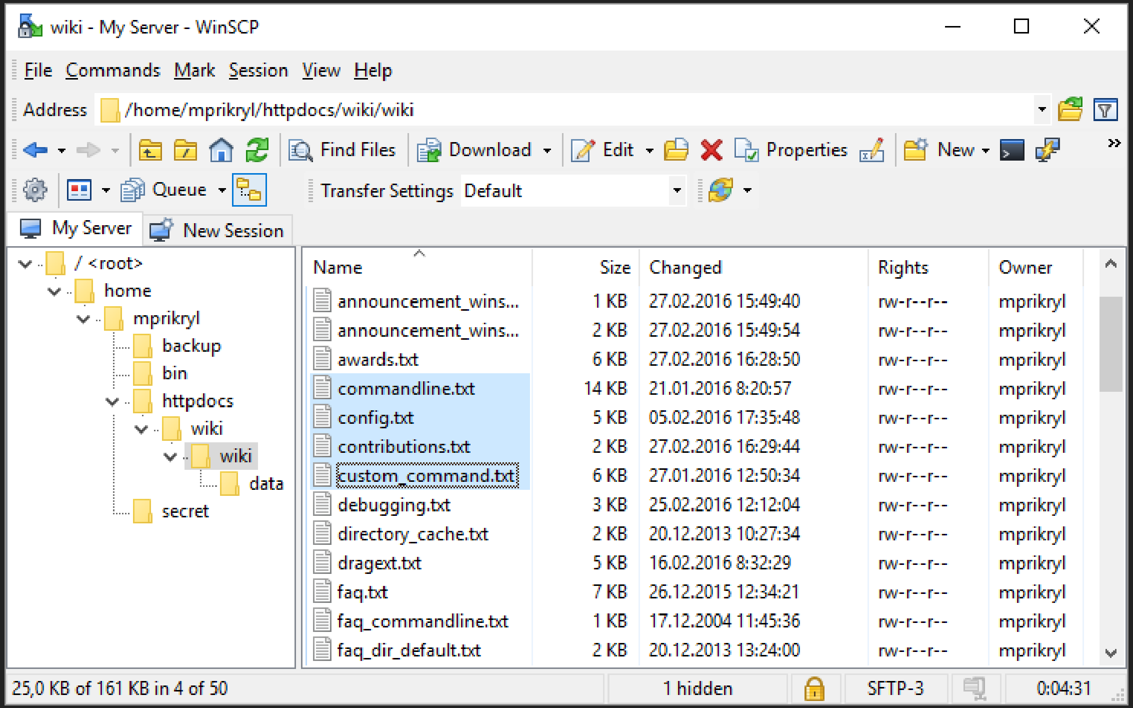
- Command line based: This is the `scp` command on Ubuntu and macOS, for Windows PuTTY users you will have `pscp`. They work like `cp` except that they can transfer files to and from a remote host over SSH.
- GUI based: options include Finder and file browser for Ubuntu and macOS, you can connect to a server. For Windows we have WinSCP or a web based environment like Jupyter Lab would also work.

```
scp mylocalfile.txt [USERNAME@]sftp.doc.ic.ac.uk:~/
```

The above command will copy the local text file to your remote home folder. The source and destination for remote addresses work the same for SSH hostnames, following `:` the path to the remote folder in this case the home folder `~/`

```
scp sftp.doc.ic.ac.uk:~/myremotefile.txt .
```

Copies the remote text file back to the current local directory `.`



Address /home/mprikryl/httpdocs/wiki/wiki

Transfer Settings Default

Name	Size	Changed	Rights	Owner
announcement_wins...	1 KB	27.02.2016 15:49:40	rw-r--r--	mprikryl
announcement_wins...	2 KB	27.02.2016 15:49:54	rw-r--r--	mprikryl
awards.txt	6 KB	27.02.2016 16:28:50	rw-r--r--	mprikryl
commandline.txt	14 KB	21.01.2016 8:20:57	rw-r--r--	mprikryl
config.txt	5 KB	05.02.2016 17:35:48	rw-r--r--	mprikryl
contributions.txt	2 KB	27.02.2016 16:29:44	rw-r--r--	mprikryl
custom_command.txt	6 KB	27.01.2016 12:50:34	rw-r--r--	mprikryl
debugging.txt	3 KB	25.02.2016 12:12:04	rw-r--r--	mprikryl
directory_cache.txt	2 KB	20.12.2013 10:27:34	rw-r--r--	mprikryl
dragext.txt	5 KB	16.02.2016 8:32:29	rw-r--r--	mprikryl
faq.txt	7 KB	26.12.2015 12:34:21	rw-r--r--	mprikryl
faq_commandline.txt	1 KB	17.12.2004 11:45:36	rw-r--r--	mprikryl
faq_dir_default.txt	2 KB	20.12.2013 13:24:00	rw-r--r--	mprikryl

25,0 KB of 161 KB in 4 of 50 | 1 hidden | SFTP-3 | 0:04:31

The graphical user interface of WinSCP.

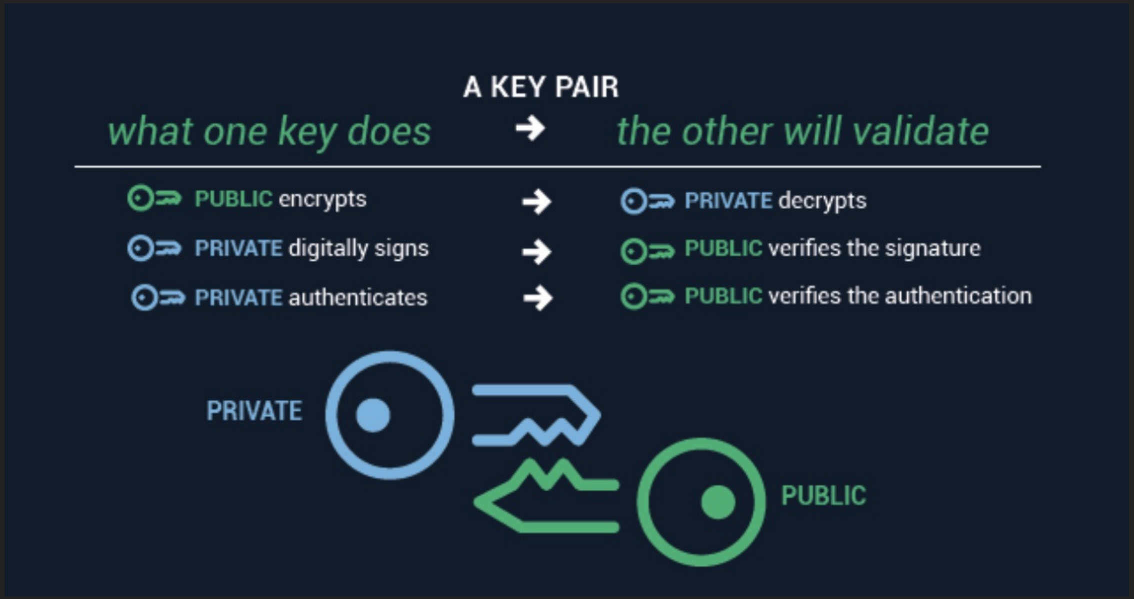
Video Walkthrough: Secure Copy

LINK

SSH Keys

So far we have been entering our username and password every time we wanted to connect over SSH. Some tools like PuTTY can try to save your password but there is a more universal and safe way of authenticating without entering passwords. This is called public key authentication and it uses public key cryptography. This is the same principle behind website certificates using HTTPS. The idea is:

1. We generate a pair of keys, one private and one public.
2. We distribute the public key to the remote machine we would like to login using our password.
3. Next time we login, the server challenges us with our public key and we validate it using our private one.
4. We're in.



A KEY PAIR

what one key does	→	*the other will validate*
PUBLIC encrypts	→	PRIVATE decrypts
PRIVATE digitally signs	→	PUBLIC verifies the signature
PRIVATE authenticates	→	PUBLIC verifies the authentication

In our case, this is to validate our identity and authenticate. You must keep your private key, ehemm, private, do not share to anyone or even across machines you own.

On Ubuntu or macOS:

1. Run `ssh-keygen` and follow the steps (accepting the defaults should be enough), you can add a password to your private key but leave it blank if you don't want to enter another password every time.
2. Run `ssh-copy-id [USERNAME@]shell1.doc.ic.ac.uk` to copy your public key to college home folder. It will essentially add your public key as a new line to `~/.ssh/authorized_keys` in your home folder.
3. Try connecting through `ssh [USERNAME@]shell1.doc.ic.ac.uk` again and you should not be asked for a password.

If you have multiple machines, generate keys for each one for better security. *You can later use the same public key for other services such as Git too.*

On Windows with PuTTY:

1. Run `PuTTYgen` program (you should have installed it with PuTTY) which is the key generator for PuTTY. It gives you a nice interface with a big button called "Generate".
2. Save the private key and public key as files on your local machine (somewhere secure, such as your home folder on your local machine).
3. Manually copy the generated public key by any secure means (scp, WinSCP) etc. to your college home folder under `~/.ssh/authorized_keys`
4. `~/.ssh/authorized_keys` is a text file with 1 public key per line. So if you have 1 public key, it's okay to overwrite this file, otherwise append to it.
5. When connecting again, under SSH select the Auth tab on the left, select the private key file you saved.

Video Walkthrough: SSH Keys

LINK

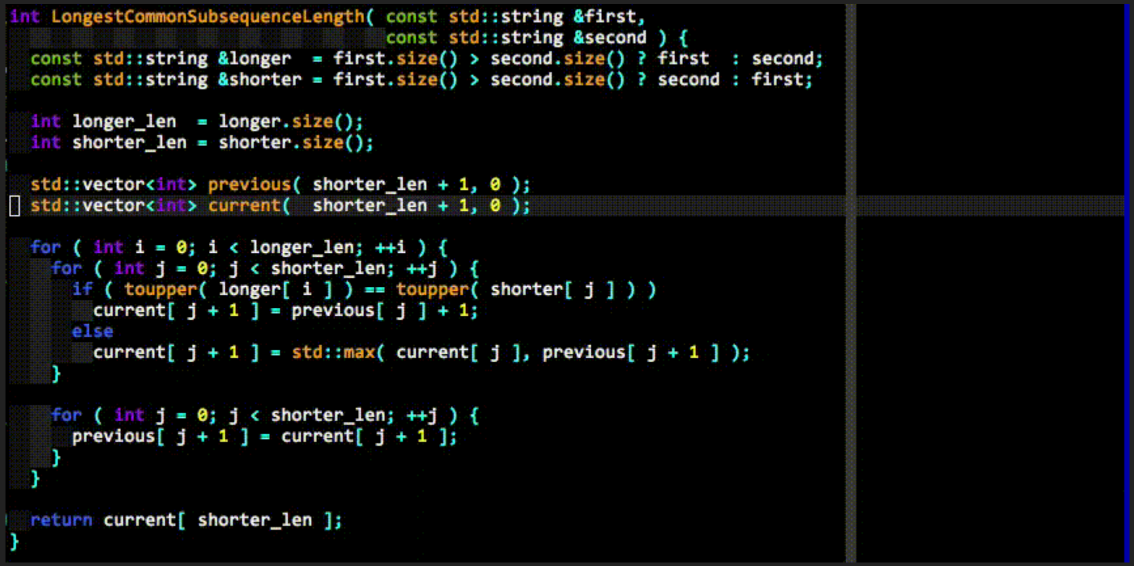
Editing Files

We ultimately want to be able to edit files to work. Mainly editing code and running it. There are three main options:

- **Command line based:** these are text based editors and live inside the terminal, no user interface. Fits perfectly within the remote terminal lifestyle. Two main contenders are VIM and Emacs. It's a huge debate on which is better, so you have to research, try and decide which one you like. For example, I'm a VIM user.
- **GUI based:** recently many GUI based editors emerged, you can try using them but they are not well suited for remote working. We can use them through X-Window-Forwarding, i.e. we forward the UI to our machine but the program runs remotely.
- **Web based:** These run inside a browser. A popular web based environment is called Jupyter Lab.

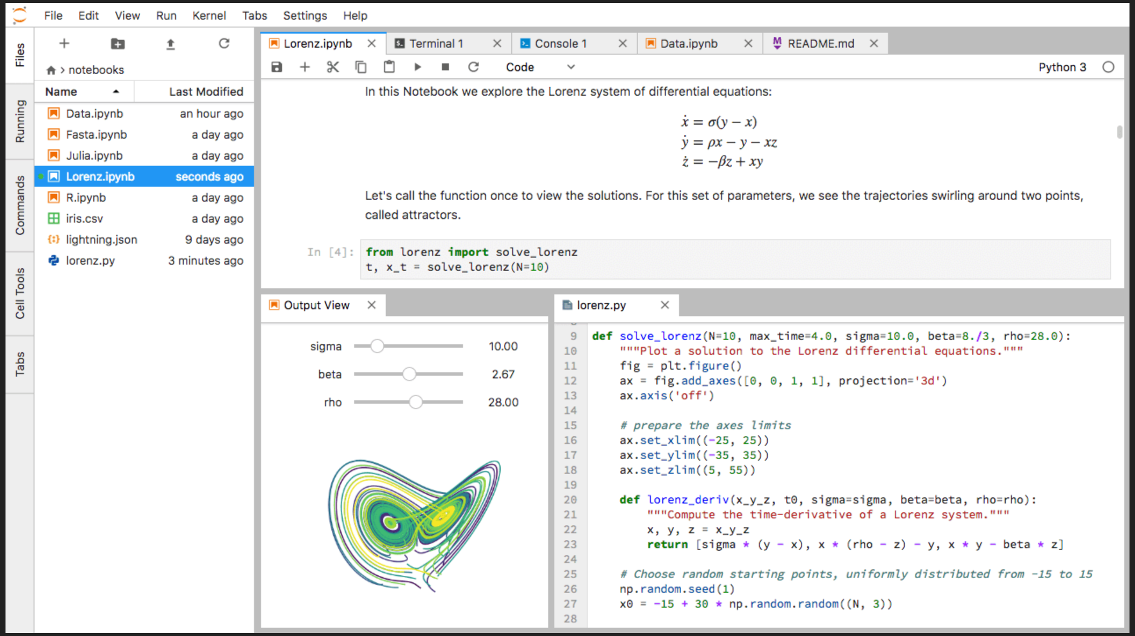
> I would recommend getting used to at least one command line editor. They are very practical, fast and can be extended using plugins.

> Take your time with editors, explore but pick one and get very comfortable with it, you'll be writing a lot of code..



```
int LongestCommonSubsequenceLength( const std::string &first,
                                    const std::string &second ) {
  const std::string &longer  = first.size() > second.size() ? first  : second;
  const std::string &shorter = first.size() > second.size() ? second : first;

  int longer_len  = longer.size();
  int shorter_len = shorter.size();

  std::vector<int> previous( shorter_len + 1, 0 );
  std::vector<int> current(  shorter_len + 1, 0 );

  for ( int i = 0; i < longer_len; ++i ) {
    for ( int j = 0; j < shorter_len; ++j ) {
      if ( toupper( longer[ i ] ) == toupper( shorter[ j ] ) )
        current[ j + 1 ] = previous[ j ] + 1;
      else
        current[ j + 1 ] = std::max( current[ j ], previous[ j + 1 ] );
    }

    for ( int j = 0; j < shorter_len; ++j ) {
      previous[ j + 1 ] = current[ j + 1 ];
    }
  }

  return current[ shorter_len ];
}
```

VIM in action. It is purely text based, fast and efficient. Connect to a machine and run `vim` or `vim [filename]` to open the program. For an interactive tutorial you can visit https://www.openvim.com/.



Name	Last Modified
Data.ipynb	an hour ago
Fasta.ipynb	a day ago
Julia.ipynb	a day ago
Lorenz.ipynb	seconds ago
R.ipynb	a day ago
iris.csv	a day ago
lightning.json	9 days ago
lorenz.py	3 minutes ago

In this Notebook we explore the Lorenz system of differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x)\\ \dot{y} &= \rho x - y - xz\\ \dot{z} &= -\beta z + xy\end{aligned}$$

Let's call the function once to view the solutions. For this set of parameters, we see the trajectories swirling around two points, called attractors.

```
In [4]: from lorenz import solve_lorenz
        t, x_t = solve_lorenz(N=10)
```

```
sigma  10.00
beta    2.67
rho    28.00
```

```python
def solve_lorenz(N=10, max_time=4.0, sigma=10.0, beta=8./3, rho=28.0):
    """Plot a solution to the Lorenz differential equations."""
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.axis('off')

    # prepare the axes limits
    ax.set_xlim((-25, 25))
    ax.set_ylim((-35, 35))
    ax.set_zlim((5, 55))

    def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
        """Compute the time-derivative of a Lorenz system."""
        x, y, z = x_y_z
        return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]

    # Choose random starting points, uniformly distributed from -15 to 15
    np.random.seed(1)
    x0 = -15 + 30 * np.random.random((N, 3))

```

A typical session in Jupyter Lab, it is more user friendly and easy to get started with.

To use Jupyter Lab

1. Pick a random number between 10000 and 20000, this is a random *port* number we will use to connect through. Also pick a random shell server, shell1, shell2, ...
2. From your machine run the following, replacing PORT with the number you picked. This will connect, request a lab machine and run jupyter connecting back through the port you specified. "-g" is for graphical, "-w" is the initial workspace. Optionally pass, `-e /path/to/virtualenv` to load a custom virtual environment that includes Jupyter Lab.

```
ssh -t -L PORT:localhost:PORT shell3.doc.ic.ac.uk "/vol/linux/bin/sshtolab -g -w ~/ -p PORT"
```

3. Follow the link provided in the terminal that is of the form `http://localhost:PORT/...` into your local browser.
4. When you are done, close your browser tab and hit Ctrl+C on the terminal to kill the Jupyter Lab session!

You can follow the video walkthrough of this as well.

To forward a remote GUI application

1. Connect to shell servers using `ssh -X [USERNAME@]shell3.doc.ic.ac.uk` `sshtolab` , "-X" stands for X-Window-Forwarding. For Mac users, you can try `-Y` instead which does trusted x-window forwarding.
2. Run a GUI program such as `gedit &`, "&" puts the program in the background.
3. After a delay, the program interface should appear on your own machine, but it is running on a college lab machine.

You will most definitely experience a lag between interacting with the interface and seeing the result. This can be unpleasant so if there is an alternative (i.e. command line editor, web based) I would recommend trying those out as well.

To forward a remote GUI on Windows

1. Install a X Server, such as VcXsrv, that will display the windows that are being forwarded. Run it.
2. In PuTTY, under SSH tab, enable X-Window-Forwarding
3. Run a remote program on a lab machine, `gedit &`
4. After a noticable delay (more than 10 seconds on first connect), you should see the program on your desktop despite running on a college lab machine.

You can follow the video walkthrough of this as well.

Video Walkthrough: Editing Files

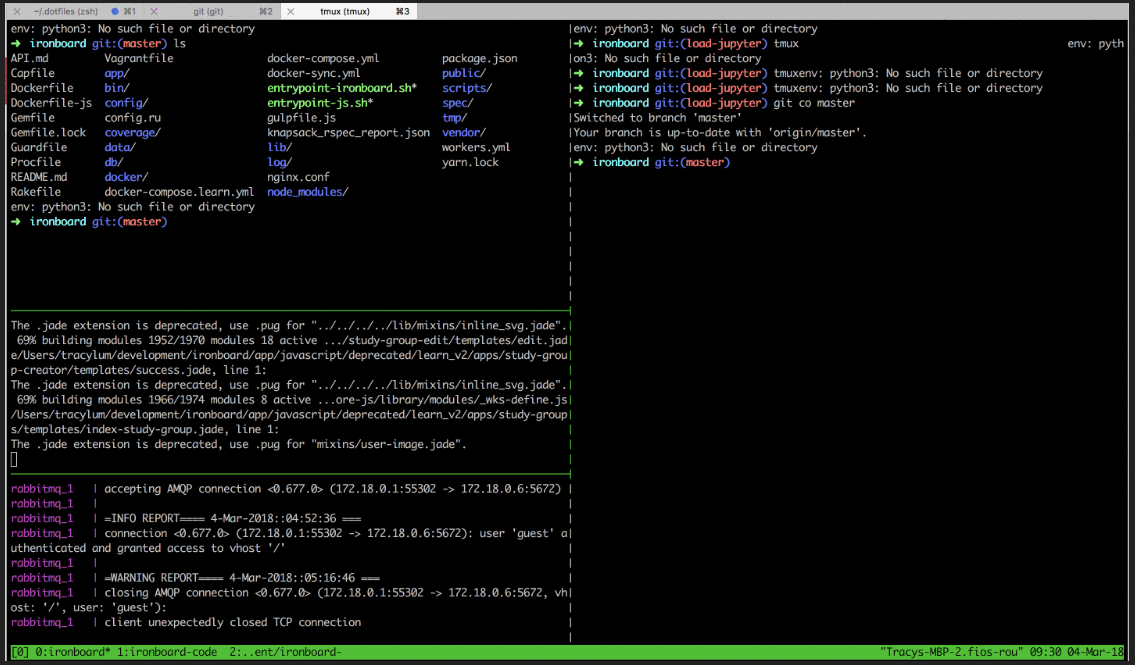
LINK

Multiple terminals (tmux)

Often we need multiple terminals: edit some files in one, run code in another etc. When you SSH into a machine you will have one terminal open. Some will try to SSH multiple times to get many terminals on the remote machine. There is a more practical and easier way: `tmux` Tmux stands for terminal multiplexer and it is a program for managing multiple terminals. You can create many windows, split them etc.

Common commands for tmux include:

- `tmux new -s [SESSION NAME]`: starts a new tmux session with the given name, you can also just run `tmux` without a name.
- `(Ctrl+B)+(Ctrl+C)`: create new window, this gives you another terminal
- `(Ctrl+B)+[0-9]`: switch to numbered window
- `(Ctrl+B)+(Ctrl+N)`: switch to next window, use P for previous window
- `(Ctrl+B)+(Ctrl+L)`: switch to the last window
- `(Ctrl+B)+%`: split window vertically, use " for horizontal split
- `(Ctrl+B)+(Ctrl+O)`: switch between window panes, the ones you split



```
env: python3: No such file or directory
➜ ironboard git:(master) ls
API.md        Vagrantfile              docker-compose.yml          package.json
Capfile       app/                     docker-sync.yml             public/
Dockerfile    bin/                     entrypoint-ironboard.sh*    scripts/
Dockerfile-js config/                  entrypoint-js.sh*           spec/
Gemfile       config.ru                gulpfile.js                 tmp/
Gemfile.lock  coverage/                knapsack_rspec_report.json  vendor/
Guardfile     data/                    lib/                        workers.yml
Procfile      db/                      log/                        yarn.lock
README.md     docker/                  nginx.conf
Rakefile      docker-compose.learn.yml node_modules/
env: python3: No such file or directory
➜ ironboard git:(master)
```

```
The .jade extension is deprecated, use .pug for "../../../../lib/mixins/inline_svg.jade".
 69% building modules 1952/1970 modules 18 active .../study-group-edit/templates/edit.jad
e/Users/tracylum/development/ironboard/app/javascript/deprecated/learn_v2/apps/study-grou
p-creator/templates/success.jade, line 1:
The .jade extension is deprecated, use .pug for "../../../../lib/mixins/inline_svg.jade".
 69% building modules 1966/1974 modules 8 active ...ore-js/library/modules/_wks-define.js
/Users/tracylum/development/ironboard/app/javascript/deprecated/learn_v2/apps/study-group
s/templates/index-study-group.jade, line 1:
The .jade extension is deprecated, use .pug for "mixins/user-image.jade".
```

```
rabbitmq_1   | accepting AMQP connection <0.677.0> (172.18.0.1:55302 -> 172.18.0.6:5672)
rabbitmq_1   |
rabbitmq_1   | =INFO REPORT==== 4-Mar-2018::04:52:36 ===
rabbitmq_1   | connection <0.677.0> (172.18.0.1:55302 -> 172.18.0.6:5672): user 'guest' a
uthenticated and granted access to vhost '/'
rabbitmq_1   |
rabbitmq_1   | =WARNING REPORT==== 4-Mar-2018::05:16:46 ===
rabbitmq_1   | closing AMQP connection <0.677.0> (172.18.0.1:55302 -> 172.18.0.6:5672, vh
ost: '/', user: 'guest'):
rabbitmq_1   | client unexpectedly closed TCP connection
```

```
env: python3: No such file or directory
➜ ironboard git:(load-jupyter) tmux                                          env: pyth
on3: No such file or directory
➜ ironboard git:(load-jupyter) tmuxenv: python3: No such file or directory
➜ ironboard git:(load-jupyter) tmuxenv: python3: No such file or directory
➜ ironboard git:(load-jupyter) git co master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
env: python3: No such file or directory
➜ ironboard git:(master)
```

```
[0] 0:ironboard* 1:ironboard-code  2:..ent/ironboard-        "Tracys-MBP-2.fios-rou" 09:30 04-Mar-18
```

A typical session inside tmux.

Video Walkthrough: Multiple terminals

LINK

Summary

These slides only include a basic starting point. We intended to give you some pointers and advice so you can work remotely effectively. I would encourage you to explore and research online more, customise the tools you use and ask questions.

Hope this was helpful and since it is new content, feedback (form link) very welcome!