ATLAS COMPUTING TUTORIAL

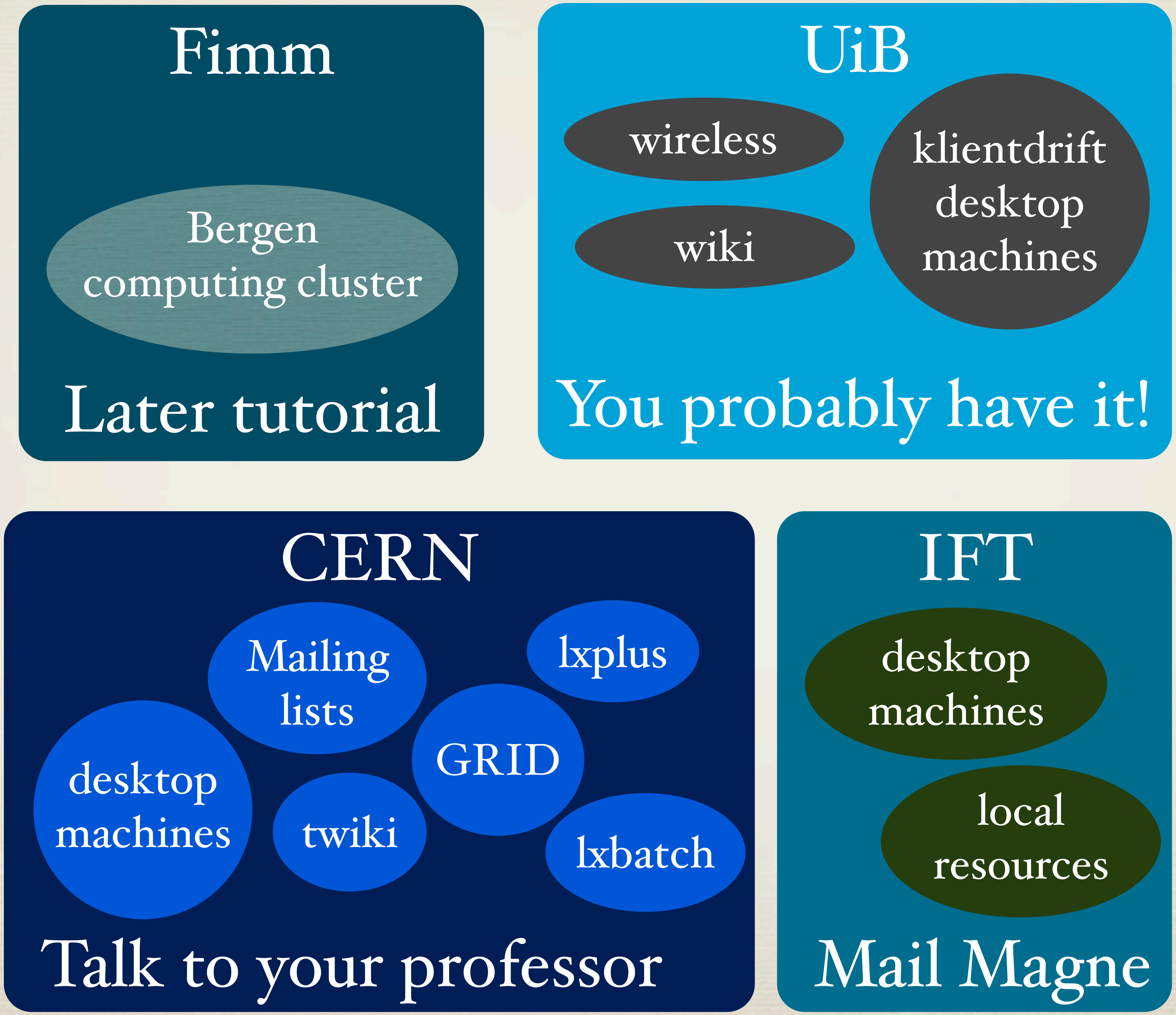
Thomas Burgess, September 2009

This is a tutorial not a talk.
If you just listen you will not learn.
Try these things out your self.
Ask questions.
Use the tutorial when you get stuck in the future

TUTORIAL OUTLINE

- Accounts and machines
- Linux shell tutorial - bash
- Writing analysis code - some ROOT / C++
- Safe and common code storage SVN

Accounts you may need



Fimm

Bergen computing cluster

Later tutorial

UiB

wireless

wiki

klientdrift desktop machines

You probably have it!

CERN

Mailing lists

lxplus

desktop machines

GRID

twiki

lxbatch

Talk to your professor

IFT

desktop machines

local resources

Mail Magne

Some of our computers

- **IFT account machines**
  - **atlasXX.ift.uib.no** - desktop machines (Scientific Linux 4)
  - **portal1.ift.uib.no & portal2.ift.uib.no**
    - external login node for iftsub machines
- **UiB IT machines**
  - **iftsubXXX.klientdrift.uib.no** - desktop machines (Fedora11)
- **CERN account machines**
  - **lxplus.cern.ch** - linux login nodes at CERN - apply (Scientific Linux 4)
- **fimm account machines**
  - **fimm.bccs.uib.no** - cluster login node at parallab (CENTOS5)

LINUX SHELL TUTORIAL

- I prefer the bash shell, if you don't have it by default, type bash - sys admin can change your default to bash
- When you open a terminal window and type in commands you are using the shell
- For remote machines log in to the shell by ssh is the only convenient way to access the machine!
- Basic understanding of the shell is very helpful in becoming an efficient linux user
- Bash is also the default shell on Mac OSX
- On windows machines you can use cygwin if you would like a proper shell

Getting a shell

- To open a shell find the command line or terminal icon, often called: kterm, gnome-terminal, xterm, iTerm, Terminal

- To open a shell on a remote machine use open a terminal and use ssh

```
ssh username@hostname
ssh -X username@hostname
```

(if you require support for graphic windows)

LINUX SHELL FILE TOOLS

man, info, whatis, mkdir, cp, scp, mv, rm, rmdir, chown, chmod, gzip, bzip2, tar, du, df

Getting help

- To get manual page

  `man top`

- To get info page (sometimes better)

  `info ssh`

- To get short help

  `whatis ls`

- To get help on a shell built in

  `help for`

- Then there is google...

mkdir - create a directory

```
mkdir tutorial
```

creates the directory “tutorial”, fails if directory exists

```
mkdir -p tutorial/dir1/dir2
```

creates all subdirectories and parent directories, doesn’t fail if directory exist

ls - list directory contents

```
ls
```

lists files in current directory

```
ls dir1/*.txt
```

lists all files ending in “.txt” in under directory “dir1”

```
ls {info,data}_{1,2,3}*
```

lists all files beginning with “info” or “data” follwed by a ”_” and the number 1 2 or 3

```
ls -lsh
```

lists in long format, sorted by size and human readable sizes

```
ls -ltr
```

lists in long format, sorted by time, reverse order

cp & mv - copy move and rename

```
cp file1 file2
```

copy “file1” to “file2”, if “file2” exists it is overwritten

```
cp -r file1 dir1/ dir2
```

copy “file1” and directory “dir1” recursively to directory “dir2” - when copying several files the last must allways be a directory!

```
cp -p file1 file2
```

copy preserving file mode, timestamps and ownership

```
scp file1 username@host:directory/
```

copy file1 to a remote machine using secure copy

```
mv file1 file2
```

move “file1” to “file2”, if “file2” exists it will be overwritten, if not “file1” will be renamed “file2”

rm & rmdir - removing things

```
rm file1 file2
```

remove “file1” and “file2”

```
rm -i *
```

remove all files - but ask for each file

```
rm -rf dir1/
```

remove directory recursively and don’t ask about anything (use with care)

```
rmdir dir1
```

remove an empty directory “dir1”

chown & chmod - manage ownership and access rights

```
chmod a+rw file1 file2
```

allow everyone to read and write to “file1” and “file2”

```
chmod -R go-rw dir
```

do not allow other users to read and write anything inside “dir”

```
chomod ug+x script.sh
```

allow you and group memebers to excecute “script.sh”

```
chown -R tburgess:atlasuib /work/atlas
```

set ownership of everything in directory to user tburgess and group atlasuib

gzip, bzip & tar - compress & archive files

```
gzip file1
```

compress “file1” to “file1.gz”

```
bzip2 file1
```

compress “file1” to “file2.bz2”

```
tar cfvj archive.tar.bz2 directory
```

compress directory to file “archive.tar.bz2”

```
tar xfvj archive.tar.bz2
```

uncompress archive

! If you want to use gzip instead, change j to z and bz2 to gz

! gzip is faster, bzip2 gives smaller files

du & df - disk usage

```
du file1
```

print disk usage of fileI

```
du -s dir
```

print summary of diskusage for directory

```
du -hs *
```

print summary of diskusage in human readable sizes for all files

```
df
```

print summary for mounted disks on the system

```
df -h /media/usb
```

print summary in human readable sizes for drive /media/usb

SYSTEM INFORMATION AND JOB CONTROL

whoami, groups, uname, hostname, top, date, pwd, jobs, fg, bg, kill, killall, ps

Some system information

- To print who is logged in, who you are, what groups you belong to, try this

```
whoami
groups
finger `whoami`
```

(note the `` which executed whoami before finger)

- To get some info about the machine try this

```
w (or who)
hostname
uname -a
hostinfo (on some systems)
cat /etc/redhat-release (on some linux systems)
top (shows most active processes)
```

Jobs and processes

- Start emacs and top sessions in terminal with &

  emacs -nw &
  top &

- The jobs are now in the background, to list them use jobs,
- To put the last job in the foreground use fg, if used with option %N (where N is the id from jobs) you can foreground any job
- control+z stops a foreground job, bg puts it in background, useful for non interactive jobs that you want to keep running
- To keep the job running even when you log off the machine use nohup command & (a log will be saved to nohup.out)
- To kill a job use kill %N, there is an numeric option -9 to kill ungracefully
- To list all your running processes use ps xu, to kill one of them use kill psid, to kill all of one process use killall processname

SHELL SCRIPTS

Environment variables, often used utilities: (echo, cat, wc, grep, sed), tests and loops

Variables

- Variables in bash can be set as followed

  ```
  variable=value
  export variable=value
  ```

  (export makes it visible outside the script)

- Variables are reffered to by putting a $ in front of its name (optional in brackets which often is useful ${variable})

  ```
  n=Thomas; s=Burgess; export myname=${n}${s}
  ```

  (semi colon is used to put several commands on the same line)

- Use “${n} ${s}” to make strings with spaces (otherwise the variable will only be ${n})

- If you use ‘${n} ${s}’ the variable names will not be printed. To print special characters use “thomas\” burgess\” \$”.

- If you use `command` the command will be executed and the result will be in the variable mydate=`date`

Some special variables

- $HOME - current users home directory (often also ~)
- $USER - current username (also username command)
- $HOSTNAME - name of host (also hostname command)
- $PWD - working directory (also pwd command)
- $PATH - colon separated list used to search for executables
- $LD_LIBRARY_PATH - colon separated list used to search for dynamic libraries
- $SHELL - name of shell (often bash or tcsh)
- $TERM - name of terminal type (often xterm)
- $RANDOM - get a random number
- $#, $0, $1-9 - number of command line arguments, name of script, argument 1 to 9

Shell Scripts

- A bash script is a text file with several lines of commands. Lines beginning with # are comments. The first line should have a special comment #!/bin/bash
- Typically bash scripts are suffixed .sh (for tcsh .csh)
- A script can be executed by

  bash script.sh
  source script.sh or . script.sh
  ./script.sh (if script is in current directoru and executable)
  script.sh (if script.sh is in PATH and executable)
- Depending on your system the script .profile or .bashrc is run everytime you start a terminal

Printing strings

- Use echo to print strings and variables

```
echo “Hello World”
echo “$SHELL in $TERM on $HOSTNAME `date`”
```

- To print to a new file use >, to append to a file use >>

```
echo “Hello new file”>file.txt
echo “Hello some more”>>file.txt
```

- To print many lines

```
echo<<EOF
line 1
line 2
EOF
```

Printing files

- To print entire contents of a file to the terminal

  ```
  cat file.txt
  ```

  (> >> works fine here also)

- To concatenate two files to one file

  ```
  cat file1.txt file2.txt > file3.txt
  ```

- To print many lines to a new file

  ```
  cat>file.txt<<EOF
  line 1
  line 2
  EOF
  ```

- To print one page at a time use more or less (less is a better version of more) instead

Count, search and replace text

- To search for a string use grep

```
grep “needle” haystack.txt
cat haystack | grep -c “needle”
```

(counts the number of needles in haystack, | sends output of previous command as input to next command)

- To count the number of words in a file, -w for number of words, -l for number of lines, -c for number of charactes

```
wc file.txt
nlines=`cat *.txt | wc -l`
```

- To replace part of string use sed

```
echo “one 2 three” | sed ‘s/2/two/’
```

- Sed and grep are very powerful commands that can do a lot more...

Tests and loops

- To check something use

  ```
  if [ $var=”one” ]; then echo “1”; fi
  if [ -e 1.txt ]; then echo “1.txt exists”; fi
  ```

- To loop use

  ```
  for x in {one, two, 3}; do echo $x; done
  for file in `ls *.txt`; do echo $file | sed ‘s/.txt//’; done
  ```

- To select from a menu use

  ```
  select num in {one,two,three}; do echo $num; break; done
  ```

  (break stops the selection once a choice is made)

ROOT

How to get started making analysis code

Setting up and starting root

- Download an appropriate root from root.cern.ch
- Unpack it somewhere and name the directory properly

  ```
  cd /scratch; tar xfvz root-5.19.4.tar.gz; mv root root-5.19.4
  ```

- Set your environment (can be in a script or in .profile / .bashrc)

  ```
  export ROOTSYS=/scratch/root-5.19.4
  export PATH=${ROOTSYS}/bin:PATH
  export LD_LIBRARY_PATH=${ROOTSYS}/lib:LD_LIBRARY_PATH
  ```

  (if using a script remember source script.sh before running)

- Start root with root -l, to start a root script directly use root -l script.C, if you want to run non interactive use root -l -q -b script.C

Root scripts

- A root macro is a textfile (usually suffixed .C) that begins with { and ends with } with lines of root commands between. Macros are executed with .x macro.C in the root prompt
- A root function is a textfile (func.C) that declares the function void func() { }. If filename and function name matches it can be executed like a macro, otherwise use .L func.C and function() in the root prompt
- Well written functions can be compiled for speed. Easiest way is to use the built in compiler in root invoked by .L func.C++
- It is also possible to use root from python, I'm no expert at this but some users prefer it to C++

```
#include <iostream>
void hello()
{
    std::cout<< "Hello world"
<< std::endl;
}

int main()
{
    hello();
}
```

Script hello.C can be run with .x, .L, compiled in root and in g++

Making a histogram with random data

```
#include “TF1.h”
#include “TH1.H”
#inclide <iostream>

void gaustest() {
TF1* f1 = new TF1("f1","1/sqrt(2*pi)*exp(-(x-5)^2/2)",0,10);
TH1* h = new TH1F("f1(x)","Gaussian test",100,0,10);
h->SetXTitle(“f1(x)”);
h->SetYTitle(“number of events”);
h->FillRandom(“f1”,5000);
h->Draw();
std::cout << "maximum = "<< h->GetMaximum() << std:: endl;
std:: cout << "max bin = " << h->GetMaximumBin() << std:: endl;
std:: cout << "max value = " << h->GetMaximumBin()*h->GetBinWidth(0) << std:: endl;
std:: cout << "Histogram mean = " << h->GetMean() << std:: endl;
std:: cout << "RMS = " << h->GetRMS() << std:: endl;
std:: cout << "Number of entries = " << h->GetEntries() << std:: endl;
h->Fit("gaus");
}
```

Converting a text file to an ntuple and plotting some data

```
cat>file.txt<<EOF
1  0.5 0.2
2  0.3 0.3
3  0.9 0.6
4  1.4 0.7
5 -1.0 0.8
6  4   1.3
EOF
```

```
Create an ntuple and read from file
root [0] TNtuple nt("ntuple","ntuple","i:x:y")
root [1] nt.ReadFile("file.txt")
Draw 1d histogram for i
root [2] nt.Draw("i”)
Draw lin style x as a function of i
root [3] nt.Draw("x:i”,””,”l”)
Draw box style for all x>0
root [4] nt.Draw("y:x”,”x>0”,”box”)
Draw x and y as a function of i in the same plot
root [5] nt.Draw("x:i”,””,”l”)
root [6] nt.SetLineStyle(2)
root [7] nt.Draw("y:i”,””,”same l”)
Draw x+y
root [8] nt.Draw("x+y:i”,””,”l”)
```

MORE ON C++ AND ROOT IN A LATER TUTORIAL...

SVN - SUBVERSION

Using a version control system to keep your source code safe!

SVN - Subversion

- Svn is used to keep files and their change history
- When developing software version control can help you
  - Change/add/remove files while maintaining file history
  - Revert or compare to an older version of a file
  - Tag project snapshots (for example a stable release)
  - Branch project for parallell development
  - Collaborate with other developers while reducing file conflicts
- Svn archives can be local, on a network or on a server.
- A properly backed up server is a good way to save files for the future.
- For ATLAS we have a svn server at CERN https://svnweb.cern.ch/trac/bergen
- More info: SVN home - http://subversion.tigris.org | Online book - http://svnbook.red-bean.com | CERN svn page - http://svn.web.cern.ch

SVN - Making a local archive

- Before starting set your svn editor in .bashrc / .profile

```
export SVN_EDITOR=emacs
```

- To learn SVN the best way is to have your own archive to play with

```
export SVNPATH=${HOME}/mysvn
mkdir -p ${SVNPATH}
svnadmin create --fs-type fsfs ${SVNPATH}
export MYSVN="file:///${SVNPATH}"
```

(SVNPATH and MYSVN are only for readability)

- To use the Bergen CERN svn instead of a local svn just change the variable

```
MYSVN=ssh+svn://svn.cern.ch/bergen
```

- For read only access (open for everyone to use)

```
MYSVN=https://svn.cern.ch/bergen
```

SVN - adding a directory

- Add a directory to your svn

  ```
  svn mkdir ${MYSVN}/directory -m "Adding directory"
  ```

  (note the comment that is inserted into your svn log. Without -m your SVN_EDITOR is started and you can edit your comment there)

- Rename directory

  ```
  svn rename ${MYSVN}/directory ${MYSVN}/dir -m “Renaming...”
  ```

- List files in your svn

  ```
  svn ls ${MYSVN}
  ```

- Remove a directory from you svn

  ```
  svn rm ${MYSVN}/dir -m "Removing directory"
  ```

- Read the revision history

  ```
  svn log ${MYSVN}
  ```

SVN - make a new project

- Create a directory for your project, and put a directory trunk/ under it - this is where your files go!

```
svn mkdir $MYSVN/myproject -m "Creating directory for my project"
svn mkdir $MYSVN/myproject -m "Creating trunk for my project"
```

- Check out the trunk of your project and change to the project directory

```
svn checkout $MYSVN/myproject/trunk myproject
cd myproject
```

(checkout can be shortened to co)

- Get project information

```
svn info
```

SVN - Importing an existing project

- Assuming you have a project with files and subdirectories
- Copy your source to a new directory

```
mkdir -p /tmp/project/trunk
cp -r project/* /tmp/project/trunk
```

- Use the import command

```
svn import /tmp/project/trunk $MYSVN -m “Initial import”
```

- Now check out trunk and continue developing there

```
svn co $MYSVN/project/trunk project-svn
```

(don’t continue in the original directory as this is not in svn)

SVN - adding a file to the project

- Make a new file and add it and commit it to the project

```
echo “Hello code” > code.txt
svn add code.txt
svn commit code.txt -m “Adding code.txt”
```

- Modify code.txt, check the svn status, list the differences

```
echo “some more code” >> code.txt
svn status
svn diff code.txt
```

- Undo any uncommitted changes to code.txt

```
svn revert code.txt
```

- Change the file again and commit the changes

```
echo “some more code” >> code.txt
svn commit
```

(will open your editor for comment & commit all changes in the directory)

- Use log to get revision history, then update to an older version, then update to the trunk

```
svn log code.txt
svn update -r 5 code.txt
svn update
```

SVN - tagging a version

- Create a directory for your tags, then copy the current version of your working directory to the svn tags directory

  ```
  svn mkdir $MYSVN/myproject/tags - “New directory for tags”
  cd ..
  svn copy myproject/ $MYSVN/myproject/tags/myproject-tag
  ```

- to check the tagged version out

  ```
  svn co $MYSVN/myproject/tags/myproject-tag myproj-tag
  ```

  (you shouldn’t commit to the tagged version, the head/trunk is for commits)

THE END