Scientific Computing (Phys 2109/Ast 3100H)
I. Scientfic Software Development

SciNet HPC Consortium

University of Toronto

November 2011

About the course

- Whole-term graduate course
- Will start with C and move to C++, also introducing Python
- Topics include scientific computing and programming skills, parallel programming, and hybrid programming.

1. Scientific Software Development: Nov 2011
   python, c, c++, git, make, modular programming, debugging
2. Numerical Tools for Physical Scientists: Jan/Feb 2012
   modelling, floating point, Monte Carlo, ODE, linear algebra,fft
3. High Performance Scientific Computing: Feb/Mar 2012
   profiling, optimization, openmp, mpi and hybrid programming

Can be taken separately by astrophysics students as mini-courses (Ast3100H), by physics students as modular courses (Phys2109).

About part I of the course

Scientific Software Development

- **Prerequisites**:
  Some programming experience. Some unix prompt experience.
- **Software that you'll need**:
  A unix-like environment with the GNU compiler suite (e.g. Cygwin), and Python (Enthought) installed on your laptop.
- **Instructors and office hours**
  Ramses van Zon - 256 McCaul Street, Rm 228 - Mon 3–4pm
  L. Jonathan Dursi - 256 McCaul Street, Rm 216 - Wed 3–4pm
- **Grading scheme**
  Four home work sets.
  To be returned by email on the next Thursday by noon.
- **Please fill out the sign-up sheet!**

About part I of the course

Scientific Software Development

Roadmap

Lecture 1	C intro, make, version control (git)
Lecture 2	PDEs, modular programming, refactoring, and testing, visualization w/python
Lecture 3	Structures in C, ODE, interpolation.
Lecture 4	C++, python, profiling

Part I

Introduction to Software Development

This lecture...

C Introduction

- Language
- Libraries
- Compilation

Version Control

- Theory
- Git
- Examples

Hands-on

C intro: Basics

- C was designed for (unix) system programming.
- C has a very small base.
- Most functionality is in (standard) libraries.
- We will use C99.

C intro: Basics

- C was designed for (unix) system programming.
- C has a very small base.
- Most functionality is in (standard) libraries.
- We will use C99.

Example (Basic C program)

```c
#include <stdio.h>
//include stdio.h to print
int main() //always called first
{ //braces delimit code block
    printf("Hello world.\n");
    //function call to print
    //line ends with a semicolon
    return 0;
    //return value to shell
}
```

C intro: Basics

- C was designed for (unix) system programming.
- C has a very small base.
- Most functionality is in (standard) libraries.
- We will use C99.

Example (Basic C program)

```c
#include <stdio.h>
//include stdio.h to print
int main() //always called first
{ //braces delimit code block
    printf("Hello world.\n");
    //function call to print
    //line ends with a semicolon
    return 0;
    //return value to shell
}
```

```
$ gcc -o hello hello.c -std=c99
 {-O2
  -Os
  -O3
  -Ofast
$
```

C intro: Basics

- C was designed for (unix) system programming.
- C has a very small base.
- Most functionality is in (standard) libraries.
- We will use C99.

Example (Basic C program)

```c
#include <stdio.h>
//include stdio.h to print
int main() //always called first
{ //braces delimit code block
   printf("Hello world.\n");
   //function call to print
   //line ends with a semicolon
   return 0;
   //return value to shell
}
```

```
$ gcc -o hello hello.c -std=c99
 ⎧-O2
 ⎨-Os
 ⎨-O3
 ⎩-Ofast
$ ./hello
Hello world.
$
```

C intro: Basics

- C was designed for (unix) system programming.
- C has a very small base.
- Most functionality is in (standard) libraries.
- We will use C99.

Example (Basic C program)

```c
#include <stdio.h>
//include stdio.h to print
int main() //always called first
{ //braces delimit code block
   printf("Hello world.\n");
   //function call to print
   //line ends with a semicolon
   return 0;
   //return value to shell
}
```

```
$ gcc -o hello hello.c -std=c99
 ⎧-O2
 ⎨-Os
 ⎪-O3
 ⎩-Ofast
$ ./hello
Hello world.
$ echo $?
0
$ █
```

C intro: Functions

Function declaration (prototype)

```
returntype name(argument-spec);
```

Function definition

```
returntype name(argument-spec) {
   statements
}
```

Function call

```
var=name(arguments);
f(name(arguments));
```

Procedures

Procedures are functions with return-type `void` ; called without assignment.

C intro: Variables

Define a variable with

```
type name [= value];
```

- *type* may be a
  - built-in type:
    - floating point type:
      `float`, `double`, `long double`
    - integer type:
      `short`,[`unsigned`] `int`, [`unsigned`] `long int` ,[`unsigned`] `long long int`
    - character or string of characters:
      `char`, `char*`
  - array, pointer
  - structure, enumerated type, union
- Variable declarations and code may be mixed in C99.
- Variables can be initialized to a *value* when declared.
  Any non-initialized variable is not set to zero, but has a random value!

C intro: Loops

```
for (initialization; condition; increment) {
    statements
}
```

```
while (condition) {
    statements
}
```

You can use `break` to exit the loop.

C intro: Loops

```
for (initialization; condition; increment) {
    statements
}
```

```
while (condition) {
    statements
}
```

You can use `break` to exit the loop.

Example

```
#include <stdio.h>
int main() {
   for (int i=1; i<=10; i++)
      printf("%d ",i);
   // note the omitted braces
   printf("\n);
}
```

C intro: Loops

```
for (initialization; condition; increment) {
    statements
}
```

```
while (condition) {
    statements
}
```

You can use `break` to exit the loop.

Example

```c
#include <stdio.h>
int main() {
   for (int i=1; i<=10; i++)
      printf("%d ",i);
   // note the omitted braces
   printf("\n");
}
```

```
$ gcc -o count count.c -O2 -
std=c99
$ ./count
1 2 3 4 5 5 6 7 8 9 10
$ █
```

C intro: Pointers

```
type *name;
```

C intro: Pointers

type ∗*name*;

Example (Pointer assignment)

```
#include <stdio.h>
int main() {
   int a=7,b=5;
   int *ptr=&a;
   a = 13;
   b = *ptr;
   printf("b=%d\n,b);
}
```

```
$ gcc -o ptrex ptrex.c -O2 -
std=c99
$ ./ptrex
b=13
$ █
```

C intro: Pointers

```
type *name;
```

Example (Pointer assignment)

```
#include <stdio.h>
int main() {
    int a=7,b=5;
    int *ptr=&a;
    a = 13;
    b = *ptr;
    printf("b=%d\n,b);
}
```

```
$ gcc -o ptrex ptrex.c -O2 -
std=c99
$ ./ptrex
b=13
$ █
```

Example (Pass by reference)

```
void inc(int *i)
{(*i)++;}
int main() {
    int j=10;
    inc(&j);
    return j;
}
```

```
$ gcc -o passref passref.c -O2 -
std=c99
$ ./passref
$ echo $?
11
$ █
```

C intro: Automatic arrays

```
type name[number];
```

- `name` is equivalent to a pointer to the first element.
- Usage `name[i]`. Equivalent to `*(name+i)`.
- C arrays are zero-based.

C intro: Automatic arrays

```
type name[number];
```

- *name* is equivalent to a pointer to the first element.
- Usage *name*[i]. Equivalent to *(*name*+i).
- C arrays are zero-based.

Example

```c
#include <stdio.h>
int main() {
    int
    a[10]={1,2,3,4,5,6,7,8,9,11};
    int sum=0;
    for (int i=0;i<10;i++)
        sum += a[i];
    printf("sum=%d\n,sum);
}
```

```
$ gcc -o autoarr autoarr.c -O2
 -std=c99
$ ./autoarr
56
$ 
```

C intro: Automatic arrays

```
type name[number];
```

- *name* is equivalent to a pointer to the first element.
- Usage *name*[i]. Equivalent to *(*name*+i).
- C arrays are zero-based.

Example

```
#include <stdio.h>
int main() {
    int
    a[10]={1,2,3,4,5,6,7,8,9,11};
    int sum=0;
    for (int i=0;i<10;i++)
        sum += a[i];
    printf("sum=%d\n,sum);
}
```

B A D !!

```
$ gcc -o autoarr autoarr.c -O2
 -std=c99
$ ./autoarr
56
$ █
```

C intro: Automatic arrays

```
type name[number];
```

- *name* is equivalent to a pointer to the first element.
- Usage *name*[i]. Equivalent to *(*name*+i).
- C arrays are zero-based.

Example

```
#include <stdio.h>
int main() {
    int
    a[10]={1,2,3,4,5,6,7,8,9,11};
    int sum=0;
    for (int i=0;i<10;i++)
        sum += a[i];
    printf("sum=%d\n,sum);
}
```

BAD !!

```
$ gcc -o autoarr autoarr.c -O2
 -std=c99
$ ./autoarr
56
$ █
```

Gotcha:

- There's an compiler dependent limit on *number*.
- C standard only says at least 65535 bytes.

C intro: Dynamically allocated arrays

Requires header file:

```
#include <stdlib.h>
```

Defined as a pointer to memory:

```
type *name;
```

Allocated by a function call:

```
name=malloc(number*sizeof(type));
```

Usages:

```
a=name[number];
```

Deallocated by a function call:

```
free(name);
```

- System function call can access all available memory.
- Can check if allocation failed ($\mathit{name} == 0$).
- Can control when memory is given back.

C intro: Dynamically allocated arrays

Example

C intro: Dynamically allocated arrays

Example

```
#include <stdlib.h>
#include <stdio.h>
void printarr(int n, int *a) {
    for (int i=0;i<n;i++)
        printf("%d ", a[i]);
    printf("\n");
}
int main(){
    int n=100;
    int*b=malloc(n*sizeof(*b));
    for (int i=0;i<n;i++)
        b[i]=i*i;
    printarr(n,b);
    free(b);
}
```

C intro: Dynamically allocated arrays

Example

```c
#include <stdlib.h>
#include <stdio.h>
void printarr(int n, int *a) {
   for (int i=0;i<n;i++)
      printf("%d ", a[i]);
   printf("\n");
}
int main(){
   int n=100;
   int*b=malloc(n*sizeof(*b));
   for (int i=0;i<n;i++)
      b[i]=i*i;
   printarr(n,b);
   free(b);
}
```

```
$ gcc -o dynarr dynarr.c -O2
-std=c99
$ ./dynarr
0 1 4 9 16 25 36 49 64 81 100 121
144 169 196 225 256 289 324 361
400 441 484 529 576 625 676 729
784 841 900 961 1024 1089 1156
1225 1296 1369 1444 1521 1600 1681
1764 1849 1936 2025 2116 2209 2304
2401 2500 2601 2704 2809 2916 3025
3136 3249 3364 3481 3600 3721 3844
3969 4096 4225 4356 4489 4624 4761
4900 5041 5184 5329 5476 5625 5776
5929 6084 6241 6400 6561 6724 6889
7056 7225 7396 7569 7744 7921 8100
8281 8464 8649 8836 9025 9216 9409
9604 9801
$ █
```

C intro: Conditionals

```
if (condition) {
   statements
} else if (other condition) {
   statements
} else {
   statements
}
```

Example

C intro: Conditionals

```
if (condition) {
    statements
} else if (other condition) {
    statements
} else {
    statements
}
```

Example

```
int main(){
    int n=20;
    int*b=malloc(n*sizeof(*b));
    if (b==0)
        return 1; //error
    else {
        for (int i=0;i<n;i++)
            b[i]=i*i;
        printarr(n,b);
        free(b);
    }
}
```

C intro: Conditionals

```
if (condition) {
    statements
} else if (other condition) {
    statements
} else {
    statements
}
```

Example

```
int main(){
    int n=20;
    int*b=malloc(n*sizeof(*b));
    if (b==0)
        return 1; //error
    else {
        for (int i=0;i<n;i++)
            b[i]=i*i;
        printarr(n,b);
        free(b);
    }
}
```

```
$ gcc -o ifm ifm.c -O2 -std=c99
$ ./ifm
0 1 4 9 16 25 36 49 64 81 100
121 144 169 196 225 256 289
324 361
$ █
```

C intro: Multidimensional arrays

```
#include <stdlib.h>
#include <assert.h>
float **matrix(long n,long m) {
    float **a=malloc(n*sizeof(*a));
    assert(a); //check if a not 0
    a[0]=malloc(n*m*sizeof(**a));
    assert(a[0]); //check if a[0] not 0
    for (long i=1; i<n; i++)
        a[i]=&a[0][i*m];
    return a;
}
void free_matrix(float **a) {
    free(a[0]);
    free(a);
}
void fill(long n,long m,
.           float **a,float v){
    for (long i=0; i<n; i++)
        for (long j=0; j<m; j++)
            a[i][j]=v;
}
```

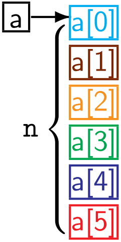
C intro: Multidimensional arrays

```
#include <stdlib.h>
#include <assert.h>
float **matrix(long n,long m) {
   float **a=malloc(n*sizeof(*a));
   assert(a); //check if a not 0
   a[0]=malloc(n*m*sizeof(**a));
   assert(a[0]); //check if a[0] not 0
   for (long i=1; i<n; i++)
      a[i]=&a[0][i*m];
   return a;
}
void free_matrix(float **a) {
   free(a[0]);
   free(a);
}
void fill(long n,long m,
.         float **a,float v){
   for (long i=0; i<n; i++)
      for (long j=0; j<m; j++)
         a[i][j]=v;
}
```

a

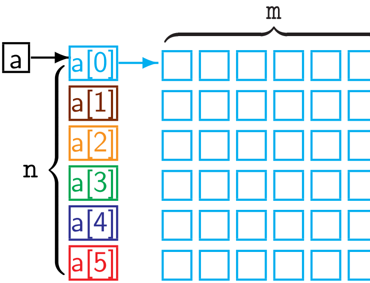
C intro: Multidimensional arrays

```
#include <stdlib.h>
#include <assert.h>
float **matrix(long n,long m) {
   float **a=malloc(n*sizeof(*a));
   assert(a); //check if a not 0
   a[0]=malloc(n*m*sizeof(**a));
   assert(a[0]); //check if a[0] not 0
   for (long i=1; i<n; i++)
      a[i]=&a[0][i*m];
   return a;
}
void free_matrix(float **a) {
   free(a[0]);
   free(a);
}
void fill(long n,long m,
.         float **a,float v){
   for (long i=0; i<n; i++)
      for (long j=0; j<m; j++)
         a[i][j]=v;
}
```



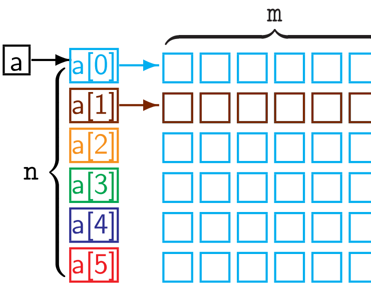
C intro: Multidimensional arrays

```
#include <stdlib.h>
#include <assert.h>
float **matrix(long n,long m) {
   float **a=malloc(n*sizeof(*a));
   assert(a); //check if a not 0
   a[0]=malloc(n*m*sizeof(**a));
   assert(a[0]); //check if a[0] not 0
   for (long i=1; i<n; i++)
      a[i]=&a[0][i*m];
   return a;
}
void free_matrix(float **a) {
   free(a[0]);
   free(a);
}
void fill(long n,long m,
.        float **a,float v){
   for (long i=0; i<n; i++)
      for (long j=0; j<m; j++)
         a[i][j]=v;
}
```



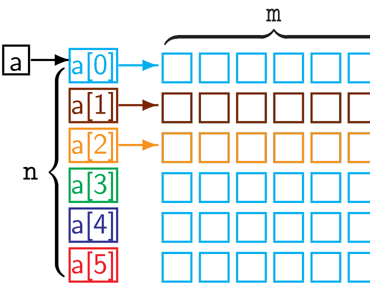
C intro: Multidimensional arrays

```
#include <stdlib.h>
#include <assert.h>
float **matrix(long n,long m) {
   float **a=malloc(n*sizeof(*a));
   assert(a); //check if a not 0
   a[0]=malloc(n*m*sizeof(**a));
   assert(a[0]); //check if a[0] not 0
   for (long i=1; i<n; i++)
      a[i]=&a[0][i*m];
   return a;
}
void free_matrix(float **a) {
   free(a[0]);
   free(a);
}
void fill(long n,long m,
.         float **a,float v){
   for (long i=0; i<n; i++)
      for (long j=0; j<m; j++)
         a[i][j]=v;
}
```



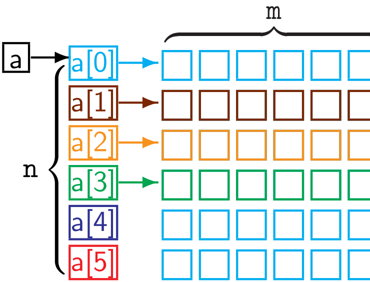
C intro: Multidimensional arrays

```
#include <stdlib.h>
#include <assert.h>
float **matrix(long n,long m) {
   float **a=malloc(n*sizeof(*a));
   assert(a); //check if a not 0
   a[0]=malloc(n*m*sizeof(**a));
   assert(a[0]); //check if a[0] not 0
   for (long i=1; i<n; i++)
      a[i]=&a[0][i*m];
   return a;
}
void free_matrix(float **a) {
   free(a[0]);
   free(a);
}
void fill(long n,long m,
.         float **a,float v){
   for (long i=0; i<n; i++)
      for (long j=0; j<m; j++)
         a[i][j]=v;
}
```



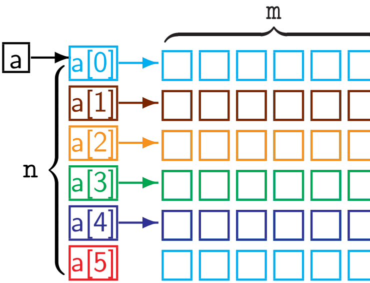
C intro: Multidimensional arrays

```c
#include <stdlib.h>
#include <assert.h>
float **matrix(long n,long m) {
    float **a=malloc(n*sizeof(*a));
    assert(a); //check if a not 0
    a[0]=malloc(n*m*sizeof(**a));
    assert(a[0]); //check if a[0] not 0
    for (long i=1; i<n; i++)
        a[i]=&a[0][i*m];
    return a;
}
void free_matrix(float **a) {
    free(a[0]);
    free(a);
}
void fill(long n,long m,
.         float **a,float v){
    for (long i=0; i<n; i++)
        for (long j=0; j<m; j++)
            a[i][j]=v;
}
```



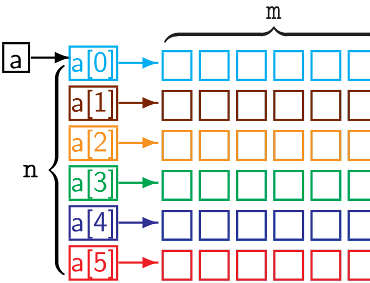
C intro: Multidimensional arrays

```
#include <stdlib.h>
#include <assert.h>
float **matrix(long n,long m) {
   float **a=malloc(n*sizeof(*a));
   assert(a); //check if a not 0
   a[0]=malloc(n*m*sizeof(**a));
   assert(a[0]); //check if a[0] not 0
   for (long i=1; i<n; i++)
      a[i]=&a[0][i*m];
   return a;
}
void free_matrix(float **a) {
   free(a[0]);
   free(a);
}
void fill(long n,long m,
.         float **a,float v){
   for (long i=0; i<n; i++)
      for (long j=0; j<m; j++)
         a[i][j]=v;
}
```



C intro: Multidimensional arrays

```c
#include <stdlib.h>
#include <assert.h>
float **matrix(long n,long m) {
   float **a=malloc(n*sizeof(*a));
   assert(a); //check if a not 0
   a[0]=malloc(n*m*sizeof(**a));
   assert(a[0]); //check if a[0] not 0
   for (long i=1; i<n; i++)
      a[i]=&a[0][i*m];
   return a;
}
void free_matrix(float **a) {
   free(a[0]);
   free(a);
}
void fill(long n,long m,
.          float **a,float v){
   for (long i=0; i<n; i++)
      for (long j=0; j<m; j++)
         a[i][j]=v;
}
```



C intro: Libraries

Usage

- Put an include line in the source code, e.g.

```
#include <stdio.h>
#include <omp.h>
#include "mpi.h"
```

- Include the libraries at link time using `-l[libname]`. Implicit for most standard libraries, with `mpicc` and `gcc -fopenmp`.

C intro: Libraries

Usage

- Put an include line in the source code, e.g.

```
#include <stdio.h>
#include <omp.h>
#include "mpi.h"
```

- Include the libraries at link time using `-l[libname]`. Implicit for most standard libraries, with `mpicc` and `gcc -fopenmp`.

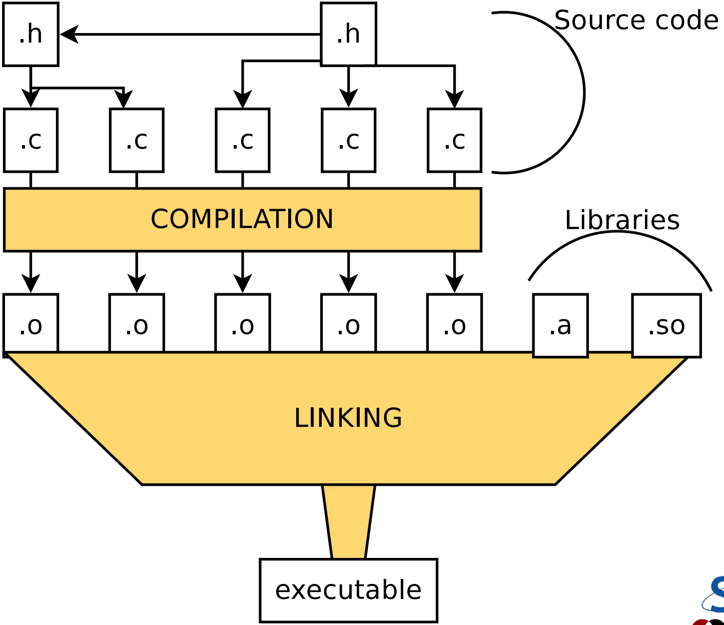
Common standard libraries

- stdio.h: input/output, e.g., `printf` and `fwrite`
- stdlib.h: memory, e.g. `malloc`
- string.h: strings, memory copies, e.g. `strcpy`
- math.h: special functions, e.g. `sqrt`. When using math, you need to link with `-lm`.

Compilation:

Building with make

Compilation workflow



Compiling with make

What does make do?

- make takes a 'makefile' and does what it specifies.
- makefile contains variables, rules and dependencies.
- makefile often called Makefile or makefile.
- if one file depends on another one that is newer, the rule is applied.
- There are default rules for e.g. c and c++ grograms.

Compiling with make

Single source file

```
# This file is called makefile
CC      = gcc
CFLAGS  = -std=c99 -O2
LDFLAGS = -lm
main:  main.c
    $(CC) $(CFLAGS) $(LDFLAGS) $^ -o $@
```

Compiling with make

Single source file

```
# This file is called makefile
CC      = gcc
CFLAGS  = -std=c99 -O2
LDFLAGS = -lm
main:  main.c
    $(CC) $(CFLAGS) $(LDFLAGS) $^ -o $@
```

Multiple source file application

```
CC      = gcc
CFLAGS  = -std=c99 -O2
LDFLAGS = -lm
main:  main.o mylib.o
    $(CC) $(LDFLAGS) $^ -o $@
main.o:  main.c mylib.h
mylib.o:  mylib.h mylib.c
clean:
    rm -f main.o mylib.o
```

Compiling with make

When typing `make` at command line:

- Checks if `main.c` or `mylib.c` or `mylib.h` were changed.
- If so, invokes corresponding rules for object files.
- Only compiles changed code files: faster recompilation.
- Parallel make:

```
$ make -j 3
```

Compiling with make

When typing `make` at command line:

- Checks if `main.c` or `mylib.c` or `mylib.h` were changed.
- If so, invokes corresponding rules for object files.
- Only compiles changed code files: faster recompilation.
- Parallel make:

```
$ make -j 3
```

Gotcha

- Make does not detect changes in compiler, or in system.
- But .o files are system/compiler dependent, so need to be recompiled.
- Always specify a “clean” rule in the makefile, so that moving from one system or compiler to another, you can do a fresh rebuild:

```
$ make clean
$ make
```

Version Control

What is it?

- A tool for managing changes in a set of files.

Version Control

What is it?

- A tool for managing changes in a set of files.
- Figuring out who broke what where and when.

Version Control

What is it?

- A tool for managing changes in a set of files.
- Figuring out who broke what where and when.

Why Do it?

- Collaboration
- Organization
- Track Changes
- Faster Development
- Reduce Errors

Collaboration

With others and yourself

Questions

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?
- What if you work on SciNet and on your own computer?

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?
- What if you work on SciNet and on your own computer?

Answers

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?
- What if you work on SciNet and on your own computer?

Answers

- Option 1: make them take turns
  - But then only one person can be working at any time
  - And how do you enforce the rule?

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?
- What if you work on SciNet and on your own computer?

Answers

- Option 1: make them take turns
  - But then only one person can be working at any time
  - And how do you enforce the rule?
- Option 2: patch up differences afterwards
  - Requires a lot of re-working
  - Stuff always gets lost

Collaboration

With others and yourself

Questions

- What if two (or more) people want to edit the same file at the same time?
- What if you work on SciNet and on your own computer?

Answers

- Option 1: make them take turns
  - But then only one person can be working at any time
  - And how do you enforce the rule?
- Option 2: patch up differences afterwards
  - Requires a lot of re-working
  - Stuff always gets lost
- Option 3: **Version Control**

Organize and Track Changes

Question

Organize and Track Changes

Question

- Want to undo changes to a file
  - Start work, realize it's the wrong approach, want to get back to starting point
  - Like "undo" in an editor...
    ...but keep the whole history of every file, forever

Organize and Track Changes

Question

- Want to undo changes to a file
    - Start work, realize it's the wrong approach, want to get back to starting point
    - Like "undo" in an editor...
      ...but keep the whole history of every file, forever
- Also want to be able to see who changed what, when
    - The best way to find out how something works is often to ask the person who wrote it

Organize and Track Changes

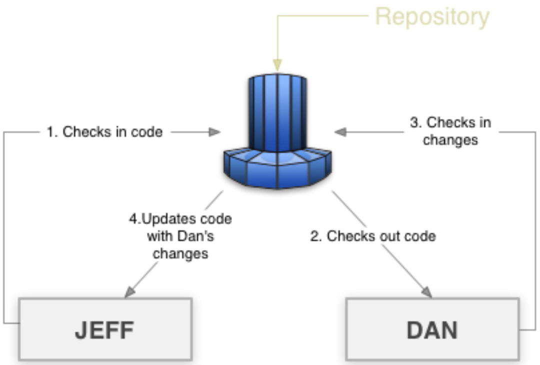
Question

- Want to undo changes to a file
  - Start work, realize it's the wrong approach, want to get back to starting point
  - Like "undo" in an editor...
    ...but keep the whole history of every file, forever
- Also want to be able to see who changed what, when
  - The best way to find out how something works is often to ask the person who wrote it

Answer

- **Version Control**

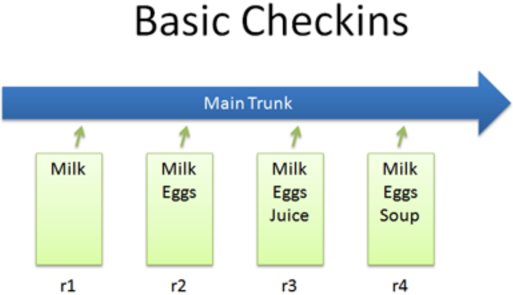
How it Works


Repository
1. Checks in code
3. Checks in changes
4.Updates code with Dan's changes
2. Checks out code
JEFF
DAN

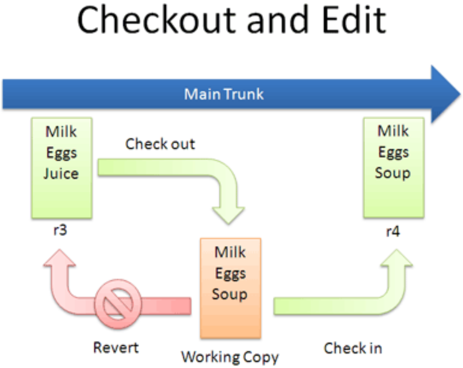
How it Works

Version Control,
Star Trek Style
revision 1
revision 2
branch 2a
branch 2b
Aieee! Roll back! Roll back!

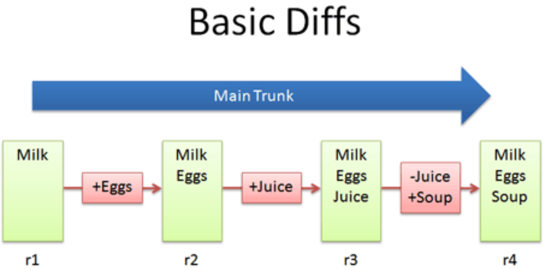
How it Works


Basic Checkins
Main Trunk
Milk
r1
Milk
Eggs
r2
Milk
Eggs
Juice
r3
Milk
Eggs
Soup
r4

How it Works


Checkout and Edit
Main Trunk
Milk
Eggs
Juice
r3
Check out
Milk
Eggs
Soup
r4
Milk
Eggs
Soup
Revert
Working Copy
Check in

How it Works



Basic Diffs

Main Trunk

Milk

+Eggs

Milk
Eggs

+Juice

Milk
Eggs
Juice

-Juice
+Soup

Milk
Eggs
Soup

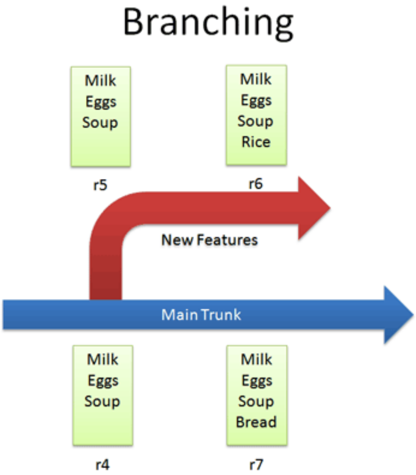
r1

r2

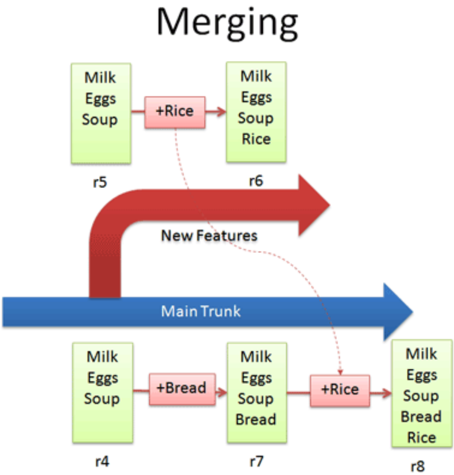
r3

r4

How it Works

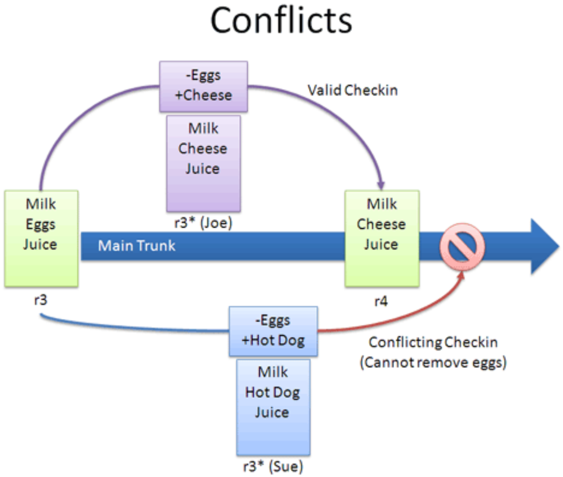

Branching
Milk
Eggs
Soup
r5
Milk
Eggs
Soup
Rice
r6
New Features
Main Trunk
Milk
Eggs
Soup
r4
Milk
Eggs
Soup
Bread
r7

How it Works



Merging
Milk
Eggs
Soup
+Rice
Milk
Eggs
Soup
Rice
r5
r6
New Features
Main Trunk
Milk
Eggs
Soup
+Bread
Milk
Eggs
Soup
Bread
+Rice
Milk
Eggs
Soup
Bread
Rice
r4
r7
r8

How it Works


Conflicts
-Eggs
+Cheese
Valid Checkin
Milk
Cheese
Juice
r3* (Joe)
Milk
Eggs
Juice
r3
Main Trunk
Milk
Cheese
Juice
r4
-Eggs
+Hot Dog
Conflicting Checkin
(Cannot remove eggs)
Milk
Hot Dog
Juice
r3* (Sue)

How it Works

Resolving Conflicts: Optimistic Concurrency

```
Milk
<<<<<<<
Cheese
=======
Hot Dog
>>>>>>>
Juice
```

What to Use

Software

- Open Source
  - Subversion, CVS, RCS
  - Git, Mercurial, Bazaar
- Commercial
  - Perforce, ClearCase

available as modules on SciNet

Software

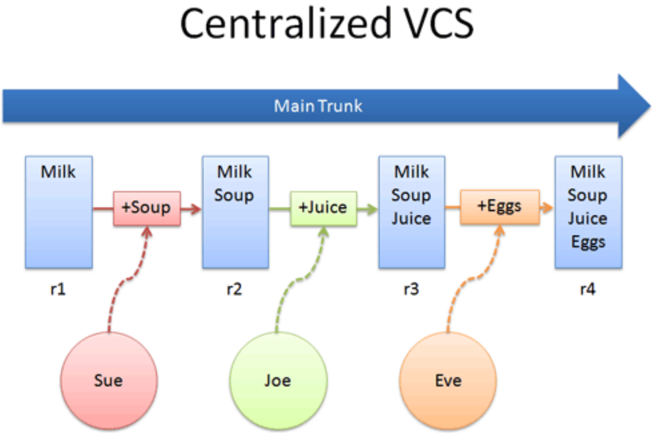
Subversion (svn)

- Centralized Version Control
- Replaces CVS
- Lots of web and GUI integration
- Users: GCC, KDE, FreeBSD

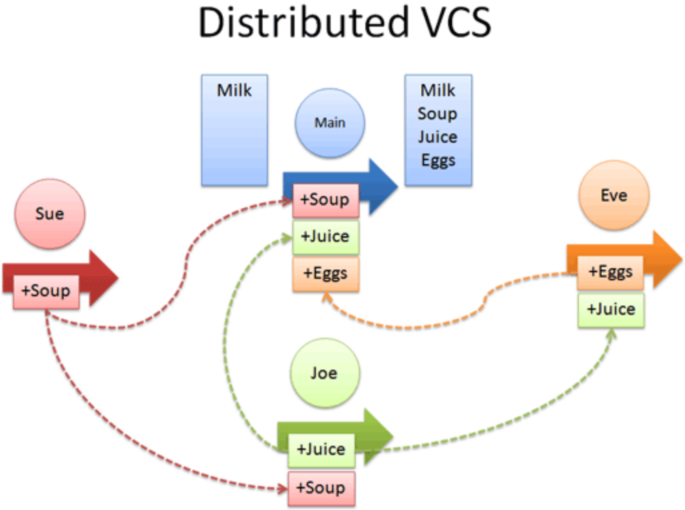
Git

- Distributed Version Control
- *nix command line driven design model
- advanced features `git-stash`, `git-rebase`, `git-cherry-pick`
- Users: Linux kernel, GNOME, Wine, X.org

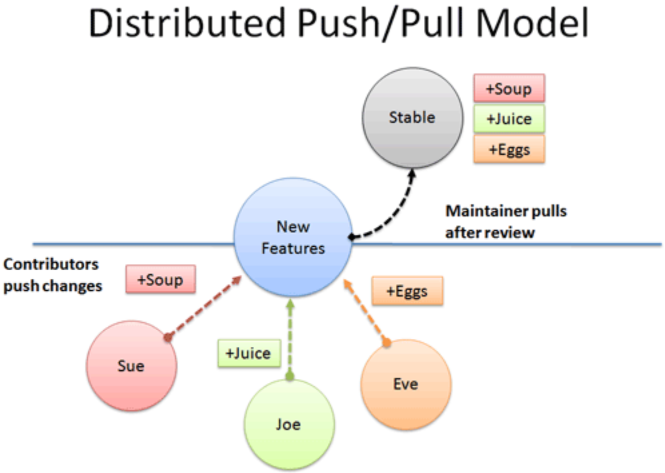
Distributed vs. Centralized



Distributed vs. Centralized



Distributed vs. Centralized



Distributed Push/Pull Model

Distributed vs. Centralized

Centralized (svn)

- Pros
  - Single Repository
  - Access Controls
  - Predictable Revision Numbers
  - GUI's
  - Simple to understand
- Cons
  - Online to access
  - Typically Slower
  - Merges can be painful

Distributed vs. Centralized

Distributed (git)

- Pros
  - Simple setup and lightweight
  - Distributed
  - Very Fast
  - Branch and merging easier
  - Sub collaboration
- Cons
  - Revision numbering
  - Can be complicated conceptually
  - Not backed up

New Repo: Git

Initialize

```
$ git config --global user.name "SciNet User"
$ git config --global user.email user@utoronto.ca
```

Create a Repository

```
$ cd ~user/code/
$ git init
$ git add .
$ git commit -m "create a git repo of my code"
```

Make Changes

```
$ vi list.txt
$ git commit -a -m "modified list.txt"
$ git log
```

Existing Repo: Git

Checkout a Project

```
$ cd ~user/code/
$ git clone /path/project/
$ git checkout master
```

Make Local Changes

```
$ vi list.txt
$ git commit -a -m "modified list.txt"
```

Publish Changes

```
$ git pull (fetch & merge)
$ git push
```

References

Links

- Git http://git-scm.com/
- Subversion http://subversion.tigris.org/

Hands-on I

Write a modular program to write “Hello, world”. Use git and make.
Start with HW1.

Homework

See hand-out.
Also: read up about basic Python, e.g. at