

# Research Computing with Python, Lecture 3, Top-down Programming and Version control

Ramses van Zon

SciNet HPC Consortium

November 12, 2013

# Top-down Programming

# About Python Programming

- In lecture 1, we saw a lot of elements of programming in python.
- We did not discuss how you actually go about using them.

# Simple example

Print the numbers from 1 to n, separated by '-', for n=1..12

1

1 - 2

1 - 2 - 3

1 - 2 - 3 - 4

1 - 2 - 3 - 4 - 5

1 - 2 - 3 - 4 - 5 - 6

1 - 2 - 3 - 4 - 5 - 6 - 7

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12

# Simple Programming Example (1)

Print the numbers from 1 to  $n$ , separated by '-', for  $n=1..12$

Forget about code, what you would do when solving this?

- 1 Take the first value of  $n$  ( $n=1$ )
- 2 Write the numbers  $1..n$
- 3 Repeat step 2 for the next value of  $n$ , unless  $n$  is equal to 12

**This is a loop!**

*This is a demonstration of top-down programming, so we'll leave the details of step 2 for later.*

# Simple Programming Example (2)

## Code sketch

```
#Print numbers from 1 to n separated by - for n=1..12  
#1. Take the first value of n (n=1)  
  
#2. Write the numbers 1..n  
  
#3. Repeat step 2 for the next value of n unless n==12
```

Doesn't matter if you do this in comments, or in some pseudo-code.

# Simple Programming Example (3)

Fill in high-level details

```
#Print numbers from 1 to n separated by - for n=1..12  
#1. Take the first value of n (n=1)  
n=1  
#2. Write the numbers 1..n  
print "Write the numbers 1 .. ",n  
#3. Repeat step 2 for next value of n unless n==12  
n=n+1  
if (n!=12) goto step_two
```

# Simple Programming Example (3)

## Oops!

- 1 Step two not really implemented, just prints what it should do.  
*Will fill in details later (top-down)*
- 2 Python does not have a goto statement.  
*We need a counting loop.*



# Intermezzo: Counting Loop

## Loops in python

Recall:

- In Python, a for loop goes over values in a list (or similar).
- Counting = going over a list of values from a start to an end.
- Such a range of values is created by the range function:

```
In [1]: range(1,13)
```

```
Out[1]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

## Writing the counting loop

```
for i in range(1,13):  
    print i, # or anything we want to do with i
```

# Intermezzo: Counting Loop

## Caution using ranges to count

- range function generates and stores all these values in memory.
- When we're counting large values, this can cause memory troubles.
- We only want one at a time
- xrange function only generates one value at a time:

```
In [2]: xrange(1,13)
```

```
Out[2]: xrange(1, 13)
```

```
In [3]: for i in xrange(1,13):
```

```
.....     print i,
```

```
.....
```

```
1 2 3 4 5 6 7 8 9 10 11 12
```

# Simple Programming Example (4)

Finish filling in high-level details

```
#Print numbers from 1 to n separated by - for n=1..12  
#1. Take the first value of n (n=1)  
for n in xrange(1,13):  
    #2. Write the numbers 1..n  
    print "Write the numbers 1 .. ",n  
#3. Repeat for next value of n unless n==12
```

Although part 2 does not do what it should yet, this does run.

# Simple Programming Example (5)

## Details for step 2

### Write the numbers 1..n

Sub-step 1: No coding, write what you would do:

- 1 Take the first value to write (1)
- 2 Write it down
- 3 Add a dash
- 4 Repeat steps 2 and 3 for next value unless it is equal to  $n$ .

# Simple Programming Example (6)

Sub-step 2: Code top level comments:

```
#Write the numbers 1..n
```

```
#1. Take the first value to write (1)
```

```
    #2. Write it down
```

```
    #3. Add a dash
```

```
#4. Repeat step 2+3 for next value unless equal to n
```

# Simple Programming Example (7)

## Sub-step 3: Fill in code

```
#Write the numbers 1..n  
#1. Take the first value to write (1)  
for i in xrange(1,n+1):  
    #2. Write it down  
    print i,  
    #3. Add a dash  
    print '-',  
#4. Repeat step 2+3 for next value unless equal to n
```

# Simple Programming Example (8)

Sub-step 4: Insert code instead of "print"

```
#Print numbers from 1 to n separated by - for n=1..12
#1. Take the first value of n (n=1)
for n in xrange(1,13):
    #2. Write the numbers 1..n
    #2.1 Take the first value to write (1)
    for i in xrange(1,n+1):
        #2.2 Write it down
        print i,
        #2.3. Add a dash
        print '-',
    #2.4. Repeat step 2+3 for next value unless equal to
#3. Repeat for next value of n unless n==12
```

Note the extra indentation!

# Simple Programming Example (9)

## Update comments

The way we do loops is a bit different from our initial layout.

```
#Print numbers from 1 to n separated by - for n=1..12
#1. Take the first value of n (n=1)
for n in xrange(1,13):
    #2. Write the numbers 1..n
    #2.1 Take the first value to write (1)
    for i in xrange(1,n+1):
        #2.2 Write it down
        print i,
        #2.3. Add a dash
        print '-',
    #2.4. Repeat step 2+3 for next value unless equal to
#3. Repeat for the next value of n unless n==12
```



# Simple Programming Example (10)

## Update comments

The way we do loops is a bit different from our initial layout:

```
#Print numbers from 1 to n separated by - for n=1..12
#1. Let n take values from 1 to 12
for n in xrange(1,13):
    #2. Write the numbers 1..n
    #2.1 Let i take values from 1 to n
    for i in xrange(1,n+1):
        #2.2 Write it down
        print i,
        #2.3. Add a dash
        print '-',
    #2.4. Repeat step 2+3 for next value of i
#3. Repeat for next value of n
```

# Simple Programming Example (11)

## Update comments

Numbering not that relevant: refer to code blocks.

```
#Print numbers from 1 to n separated by - for n=1..12
# Let n take values from 1 to 12
for n in xrange(1,13):
    # Write the numbers 1..n
    # Let i take values from 1 to n
    for i in xrange(1,n+1):
        # Write it down
        print i,
        # Add a dash
        print '-',
    # Repeat block for next value of i
# Repeat block for next value of n
```

# Simple Programming Example (12)

```
#Print numbers from 1 to n separated by - for n=1..12  
# Let n take values from 1 to 12  
for n in xrange(1,13):  
    # Write the numbers 1..n  
    # Let i take values from 1 to n  
    for i in xrange(1,n+1):  
        # Write it down  
        print i,  
        # Add a dash  
        print '-',  
        # Repeat block for next value of i  
# Repeat block for next value of n
```

## Output

```
1 - 1 - 2 - 1 - 2 - 3 - 1 - 2 - 3 - 4 - 1 - 2 - 3 - 4  
- 5 - 1 - 2 - 3 - 4 - 5 - 6 - 1 - 2 - 3 - 4 - 5 - 6 -  
7 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 1 - 2 - 3 - 4 - 5
```

# Simple Programming Example (13)

## Output

```
1 - 1 - 2 - 1 - 2 - 3 - 1 - 2 - 3 - 4 - 1 - 2 - 3 - 4
- 5 - 1 - 2 - 3 - 4 - 5 - 6 - 1 - 2 - 3 - 4 - 5 - 6 -
7 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 1 - 2 - 3 - 4 - 5
- 6 - 7 - 8 - 9 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 -
10 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 1
- 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 -
```

## What is wrong?

- No 'newlines'
- Solution: some of the dashes should be 'newlines'

# Simple Programming Example (14)

```
#Print numbers from 1 to n separated by - for n=1..12
# Let n take values from 1 to 12
for n in xrange(1,13):
    # Write the numbers 1..n
    # Let i take values from 1 to n
    for i in xrange(1,n+1):
        # Write it down
        print i,
        # Add a dash or newline
        if i<n:
            print '-',
        else:
            print ''
    # Repeat block for next value of i
# Repeat block for next value of n
```

# Simple Programming Example (15)

```
#Print numbers from 1 to n separated by - for n=1..12  
# Let n take values from 1 to 12  
for n in xrange(1,13):  
    # Write the numbers 1..n  
    # Let i take values from 1 to n-1  
    for i in xrange(1,n):  
        # Write it down  
        print i,  
        # Add a dash  
        print '-',  
    # Print value of n at the end with a newline  
    print n  
    # Repeat block for next value of i  
# Repeat block for next value of n
```

# Top-down or Bottom-up approach

## Top-down approach

- We write the top level first and then coded the lower level, i.e., writing the numbers 1..n, later.

## Bottom-up approach

- One could also first implement writing the number 1..n. Must define lower-level tools first nonetheless.
- More generally, a bottom-up approach means putting simpler pieces or code together to get a more complex system.
- In a sense, we used the bottom-up approach when we invoke the built-in functions `xrange` and `print`, which someone had already programmed for us.

# Modularity

## Modularity

- We have seen python modules that can be imported
- You can write your own modules
- Each module should have one logical function
- Each module is in its own file
- On a more basic level, organize code in functions
- Each function should have one logical function

## Potential advantages of modular programming

- More reusable than monolithic designs
- Clearer code
- Easier to maintain
- Easier to track changes



# Modularity: Example

```
# File: writenumbers.py
# Function to write a line of numbers upto n
def write_numbers_upto(n):
    """Writes the numbers 1 through the argument n"""
    # Let i take values from 1 to n-1
    for i in xrange(1,n):
        # Write it down with a dash
        print i, '-',
    # Print value of n at the end with a newline
    print n

# Function to write several lines of numbers upto m
def write_multiple_numbers_upto(m):
    """Print numbers 1 through n for n=1..m"""
    for n in xrange(1,m+1):
        write_numbers_upto(n)
```

```
import writenumbers
writenumbers.write_multiple_numbers_upto(12)
```

# Version Control (Theory)

# 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
- Reproducibility

# Collaboration

## Questions

- What if two (or more) people want to edit the same file at the same time?
- What if you work on a lab 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

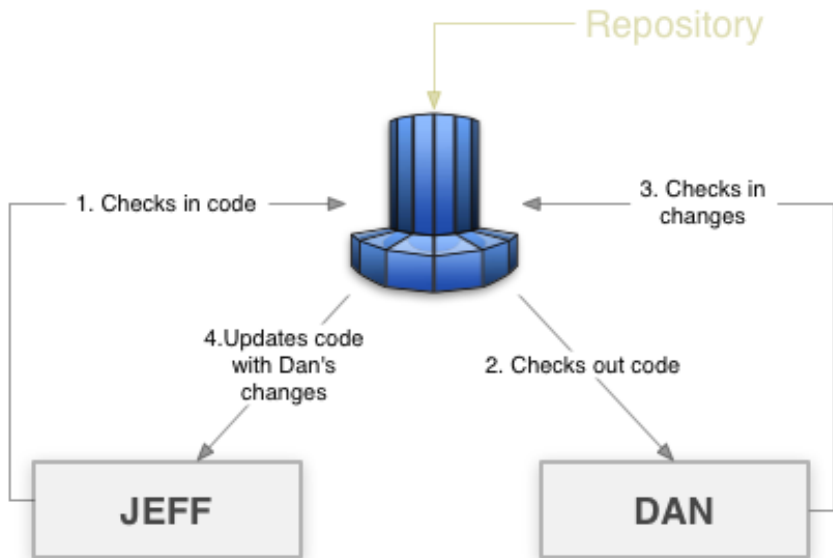
## Question

- 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

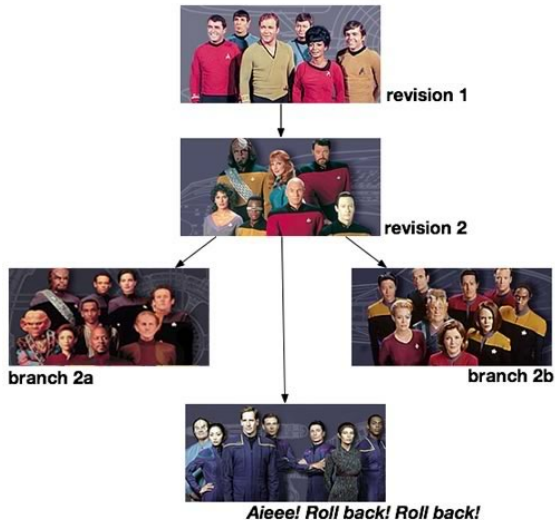
- Version Control

# How Version Control Works



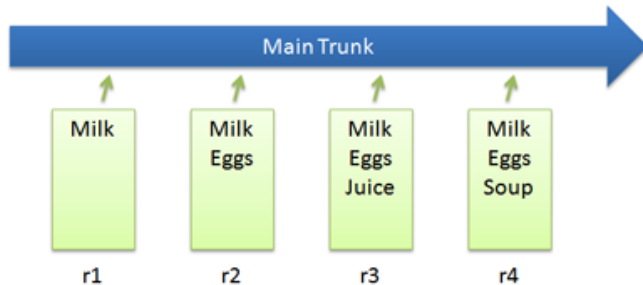
# How Version Control Works

## Version Control, *Star Trek Style*



# How Version Control Works

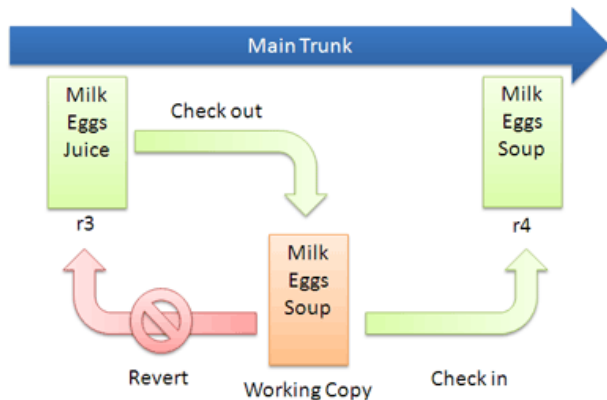
## Basic Checkins





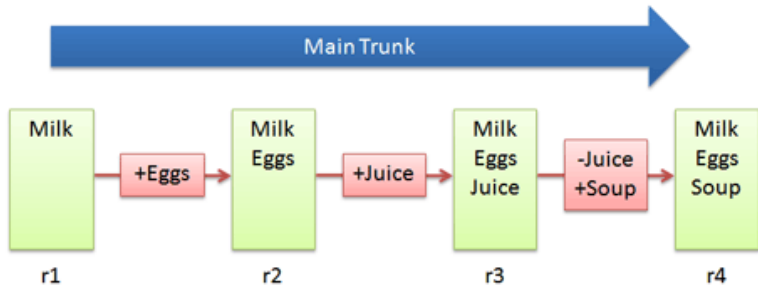
# How Version Control Works

## Checkout and Edit



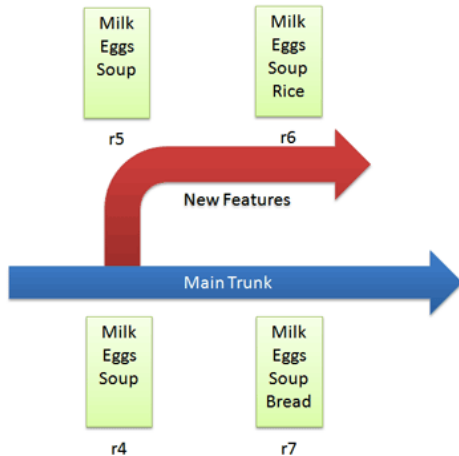
# How Version Control Works

## Basic Diffs



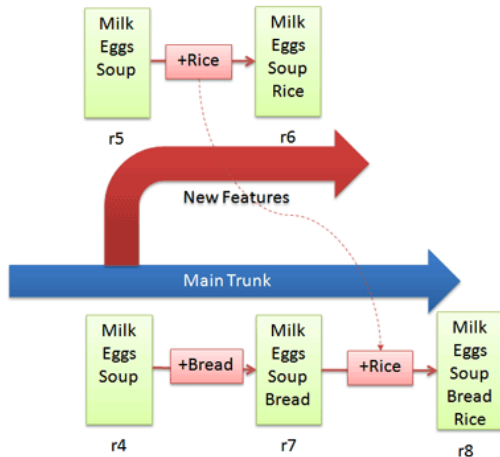
# How Version Control Works

## Branching



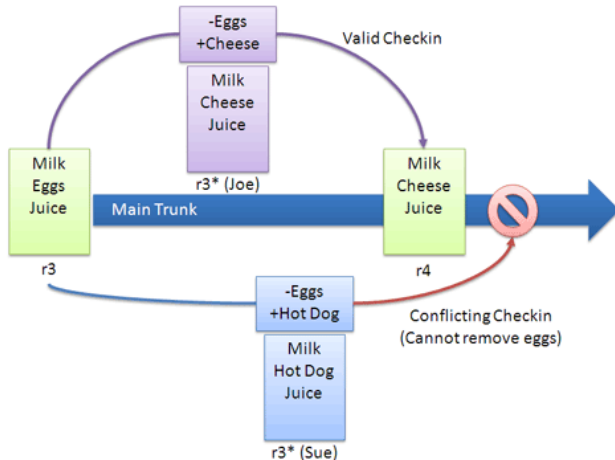
# How Version Control Works

## Merging



# How Version Control Works

## Conflicts



# How Version Control Works

## Resolving Conflicts: Optimistic Concurrency

Milk

<<<<<<<

Cheese

=====

Hot Dog

>>>>>>>

Juice

# What VC System to Use?

## Software

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

# Mercurial Version Control



# Mercurial

- Open-Source Distributed Version Control System
- Written in Python
- Relatively easy to learn
- Platform independent

# Mercurial Installation

- If you got the academic license, you can install mercurial from the Canopy package manager.
- If not, you can install it outside of Canopy and still use it. (package manager or download from mercurial site)
- After installation, tell mercurial who you are by editing  
In Windows: %HOME%/Mercurial.ini  
In Linux/Mac: \$HOME/.hgrc  
And adding

[ui]

```
username = FIRSTNAME LASTNAME <EMAILADDRESS>
```

# Mercurial Usage In Enthought Canopy

- Mercurial is used through a *command line*.
- A command can be any executable plus arguments.
- Get the OS command line from IPython with an exclamation mark.
- The name of the mercurial executable is `hg`, followed by a subcommand and other arguments.
- In IPython, mercurial commands would have form

```
!hg subcommand ...
```

- We can omit the exclamation mark if we first type

```
alias hg hg
```

# Basic things you want to do with Mercurial

- 1 Start a repository
- 2 Add files to be tracked
- 3 Store files
- 4 Modify/remove files
- 5 View (file) history
- 6 Revert changes
- 7 Copy a repository

# 1 Start a repository

```
hg init
```

This start a repository in the current directory by adding a subdirectory `.hg`.

Make sure you are in the right directory (`cd`)

## Example

```
In [1]: alias hg hg
```

```
In [2]: cd mypythonfolder
```

```
In [3]: hg init
```

## 2 Add files to be tracked

```
hg add [FILE [FILE ...]]
```

This adds FILE to be tracked in the repository.

Does not yet store it, just flags it.

If no FILEs are given adds all files in current directory.

*Note on notation: All-caps words to be replaced by real names. denotes optional arguments.*

### Example

```
In [4]: open("groceries", "w").write("Milk\n")
```

```
In [5]: hg add groceries
```

# 3 Store files

```
hg commit -m 'MESSAGE'
```

This stores all the tracked files (only changes are stored).

To view what changes haven't been committed yet:

```
hg status
```

## Example

```
In [6]: hg commit -m 'First commit of grocery list'  
groceries  
committed changeset 0:f741ddf70ab2
```

## 4 Modify/remove files

Tracked modified files are stored in the repository when you commit.

Removing tracked files will flag them as 'missing'.

To actually delete missing files from the repository:

```
hg remove -A
```

### Example

```
In [7]: open("groceries", "a").write("Eggs")
```

```
In [8]: hg status
```

```
M groceries
```

```
In [9]: hg commit -m 'Also need eggs'
```

```
groceries
```

```
committed changeset 1:44f60c750dc1
```



# 5 View (file) change history

```
hg log [FILE]
```

Shows history.

```
hg diff [FILE]
```

What changed since last commit?

## Example

```
In [10]: hg log
changeset: 1:44f60c750dc1
tag:       tip
user:      Ramses "van Zon" <rzon@scinethpc.ca>
date:      Tue Nov 12 01:15:04 2013 -0500
files:     groceries
description:
Also need eggs
```

```
changeset: 0:f741ddf70ab2
user:      Ramses "van Zon" <rzon@scinethpc.ca>
date:      Tue Nov 12 01:11:56 2013 -0500
```

## 6 Revert changes

To revert uncommitted changes

```
hg revert FILE
```

or for all files:

```
hg revert --all
```

To revert last commit

```
hg rollback
```

# 7 Copy a repository

```
hg clone SOURCEPATH TARGETPATH
```

Can be used to revert to an arbitrary version as well:

```
hg clone -r VERSION SOURCEPATH TARGETPATH
```

## Example

```
In [11]: cd ..
```

```
In [12]: hg clone mpythonfolder newpythonfolder  
updating to branch default
```

```
resolving manifests
```

```
getting groceries
```

```
1 files updated, 0 files merged, 0 files removed, 0 files
```

# A few VC tips

- Use it! Will save you trouble.
- Commit often
- Give sensible comment messages
- Don't commit derivative stuff (log files, executables, compiled python modules)
- Rarely need to commit large binary files

# Basic Mercurial Commands - Recap

---

<b>command</b>	<b>purpose</b>
hg init	create .hg repo folder for version data
hg add	add file(s) to be tracked
hg remove -A	remove already deleted files from repo
hg commit	store tracked files
hg revert	forget changes since last commit
hg rollback	forget last commit
hg clone	copy repository
hg log	history
hg diff	differences

---

## Next Time

# Next Lecture

*Thursday November 14, 2013, 11:00 am*

**Topic: Numpy and Visualization**