

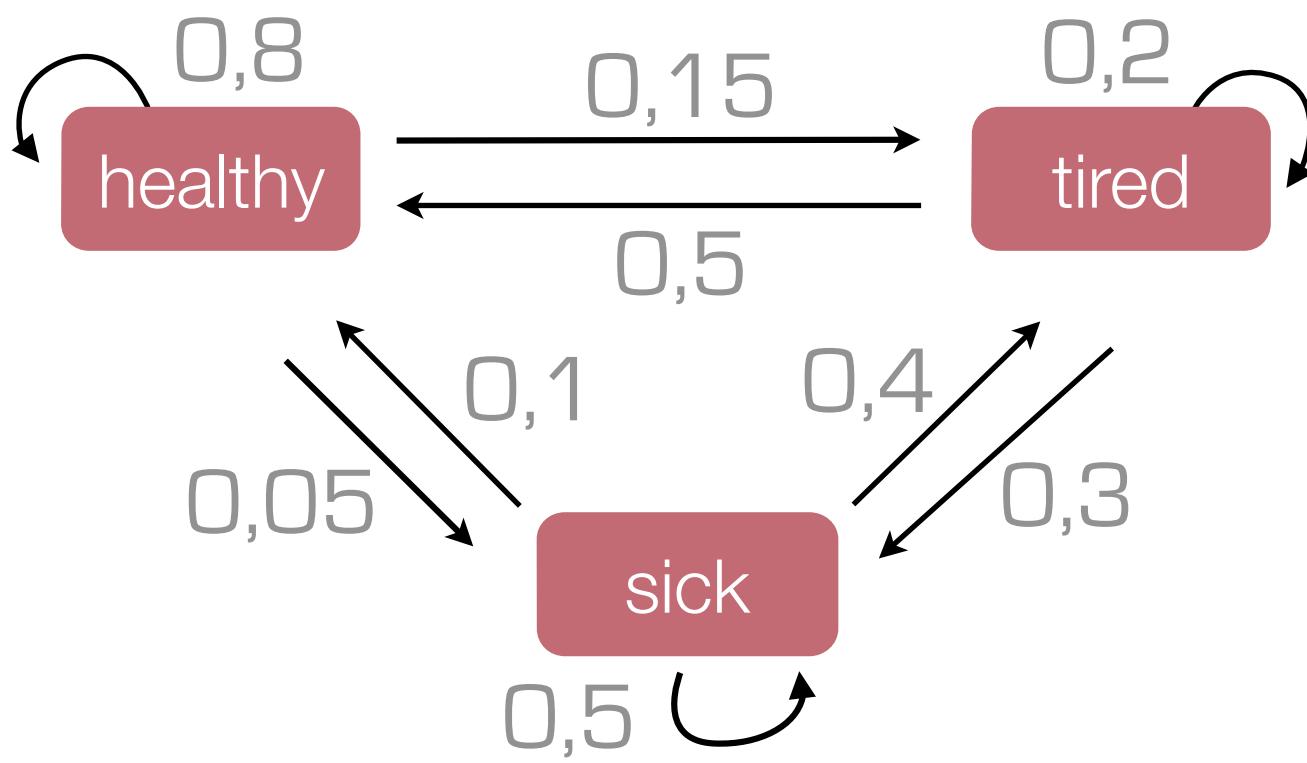
# Hidden Markov Models

---

Thierry Parmentelat — INRIA

[thierry.parmentelat@inria.fr](mailto:thierry.parmentelat@inria.fr)

# Markov Models



- model :  $\lambda=(A,\pi)$
- A: Transitions
- $\pi$ :  $p(\text{initial state}=i)$

0,8	0,15	0,05	$\Sigma=1$
0,5	0,2	0,3	
0,1	0,4	0,5	

A

0,7	0,2	0,1
-----	-----	-----

$\pi$

# Markov Models

---

- Markov hypothesis
  - behaviour depends only on current state (**not** on history)
- Observation
  - $E_1 E_2 \dots E_t \dots$  : sequence of states,  $E_i$  in  $\{1..N\}$
- Problems
  - (I) Probability of a given sequence
  - (II) Probability to observe state  $i$  at time  $t$

# Notations

---

- Mathematical:  $A_{ij}$
- Computer Science:  $A[i][j]$
- Example Problem (I)
  - Maths: sequence  $E_1 E_2 \dots E_t$   
$$\text{proba} = \pi(E_1) * \prod_{i=2..t} A_{(E_{i-1})(E_i)}$$
  - CS: sequence  $\text{seq}[t]$   
$$\text{proba} = \prod_{i=2..t} (A[\text{seq}[i-1]] [\text{seq}[i]])$$

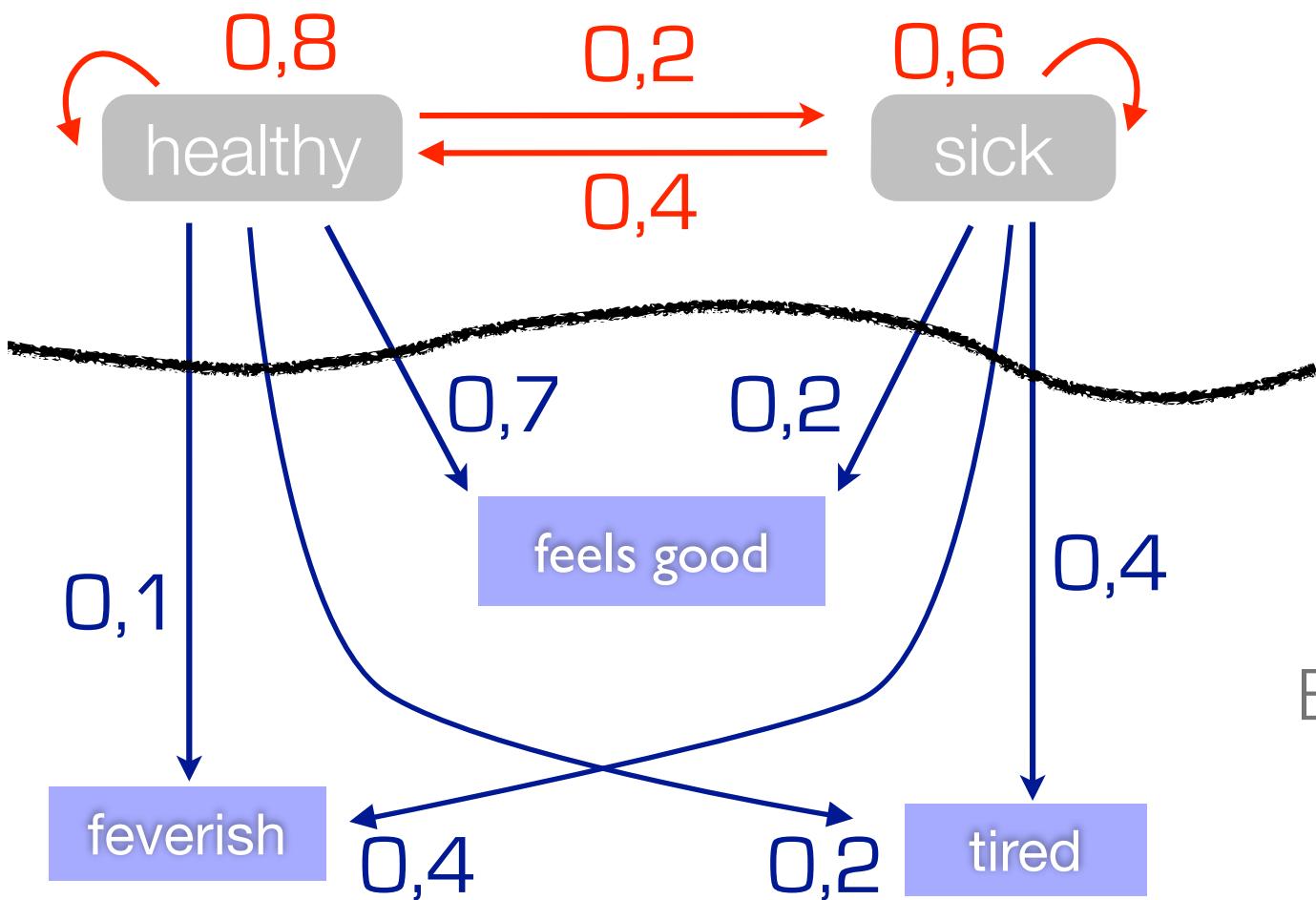
From now on,  
we will have  
all indices start at 0

# Python features, and modules of interest

---

- range                           $\text{range}(5) \rightarrow [0, 1, 2, 3, 4]$                            $\text{range}(1,5) \rightarrow [1, 2, 3, 4]$
- [ function( $x$ ) for  $x$  in inputs ]    ou [ function( $x$ ) for  $x$  in inputs if condition( $x$ ) ]
- built-in sum                           $\text{sum}(\text{range}(5)) \rightarrow 10$
- module operator from operator import add, mul
- reduce                                   $\text{reduce}(\text{add}, \text{range}(5), 0) \rightarrow 10$      $\text{reduce}(\text{mul}, \text{range}(1,5), 1) \rightarrow 24$
- numpy    [http://wiki.scipy.org/Tentative\\_NumPy\\_Tutorial](http://wiki.scipy.org/Tentative_NumPy_Tutorial)
- matplotlib                                  <http://matplotlib.org>

# Hidden Markov Models



- Model :  $\lambda = (A, \pi, E)$
- decoupling state-output

E

0,1	0,2	0,7
0,4	0,2	0,4

$\Sigma = 1$

# Two major kinds of Hidden Markov Model

---

- Discrete: Emission among a **finite** set of observations
- Continuous: Emission in  $\mathbb{R}^d$  - second chapter

# More on notations

---

- **sequence** : a list of observations (a.k.a. **seq**)
- **path** : a list of states
- indices:
  - **i,j** : for denoting states (**N**: nb of states)
  - **t** : for denoting time (**T**: max time for a given seq/path)
  - **o** : for denoting outputs/observations (**O**: nb of signals)

# Classical problems in HMM's

---

- Reference paper - Lawrence Rabiner  
A tutorial on Hidden Markov Models...  
<http://www.cs.ubc.ca/~murphyk/Bayes/rabiner.pdf>
- (I) Probability to observe a given sequence  
algos : **forward** and **backward** (a.k.a. alpha and beta)
- (II) Decoding: estimate (most probable) sequence of states that produce a given sequence - algo: **Viterbi**
- (III) Learning: given observed sequences, and from a given HMM, find an HMM that is more likely to produce them  
algo: **Baum-Welch**

# Probability for a given sequence of observations

---

- Naive algorithm:
  - probability for observing sequence along a given path
  - sum on all state paths

$$P(\text{seq}/\text{path}) = \pi(\text{path}_0) * \prod_{t=1}^{T-1} A_{\text{path}_{t-1}, \text{path}_t} * \prod_{t=0}^{T-1} E_{\text{path}_t, \text{seq}_t}$$

$$P(\text{seq}) = \sum_{\text{path} \in N^T} P(\text{seq}/\text{path})$$

- unpractical, we have  $N^T$  paths to sum on

# Forward

---

- greedy variation
- **alpha[i][t]**  
probability to observe seq **until t** and end up in **state i**
- initialization
$$\alpha_{i,0} = \pi_i * E_{i,seq_0}$$
- for i in range(N):  
$$\text{alpha}[i][0] = P[i] * E[i][seq[0]]$$

$$\alpha_{i,0} = \pi_i * E_{i,seq_0}$$

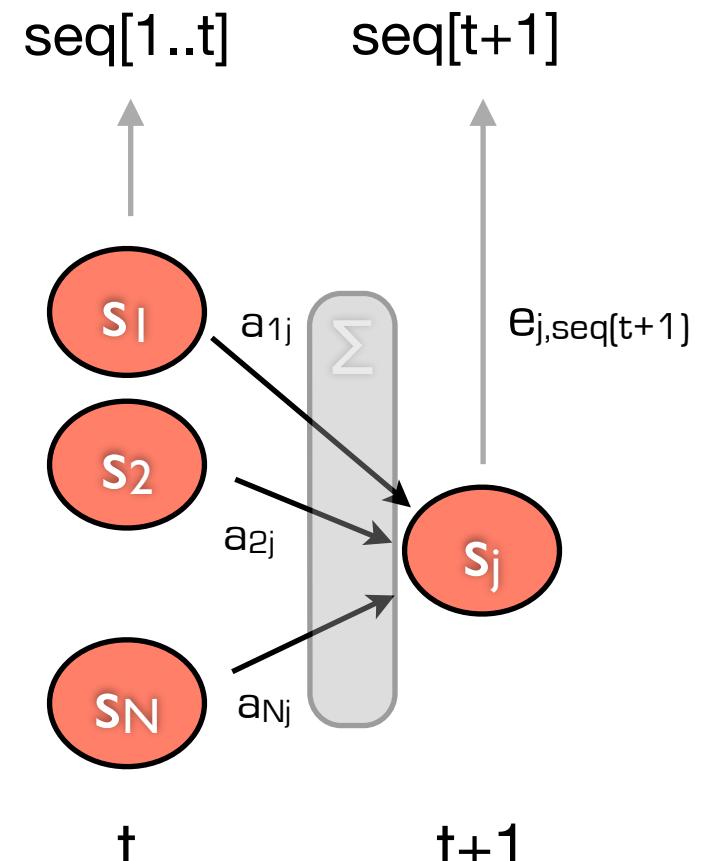
## Forward - continued

---

- A: NxN  
E: NxO  
Pi: N  
sequence: T

- Recurrence

$$\alpha_{j,t+1} = \sum_{i=0}^{N-1} \alpha_{i,t} * A_{i,j} * E_{j,seq_{t+1}}$$



- $\text{alpha}[j,t+1] = \text{reduce}(\text{add}, [\text{alpha}[i,t] * A[i,j] * E[j,seq[t+1]]] \text{ for } i \text{ in range}(N))$

# Forward - finalization

---

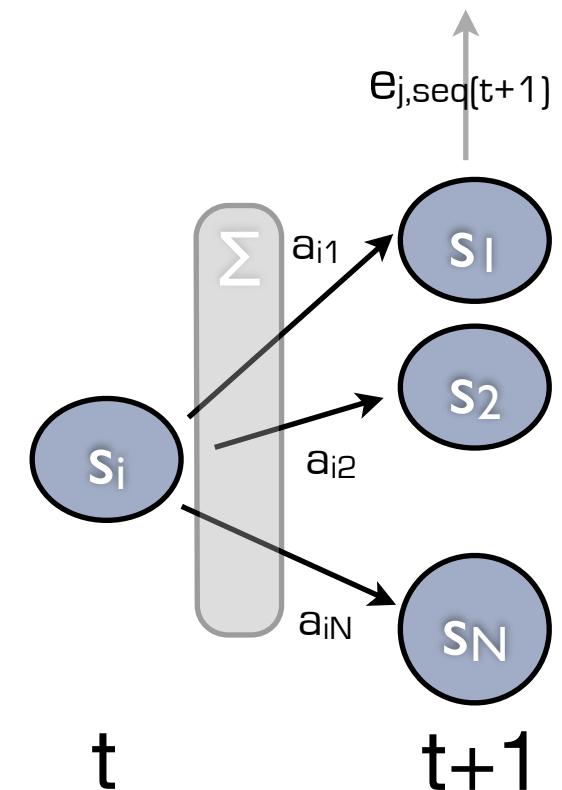
- Original purpose : compute proba (seq)
  - obtained by summing all  $\alpha[i][T]$
- Note that the details of  $\alpha$  are needed as well
  - as they will be re-used in the other algorithms
- Complexity of this algorithm
  - $\Theta(N^2 * T)$  in space and time

# Backward

---

- Same idea as *forward*, except .. the other way around
- **beta[i][t]**  
probability to observe seq **from t+1** and end up in **state i**
- $\text{beta}[i][T-1] = 1$
- $\text{beta}[i][t] = <\text{yours to say}>$
- final result

$$P(\text{seq}) = \sum_{i=0}^{N-1} \pi_i * E_{i,\text{seq}_0} * \text{beta}(i, 0)$$



# Implementation

---

- Use your locally installed python (v2 recommended ?)
- To implement a function that can be used like this  
 $(\text{proba}, \text{alpha}) = \text{forward}(\text{Pi}, \text{A}, \text{E}, \text{sequence})$
- Same for backward
- Using numpy or not - your call

# unfaircasino – a study model

---

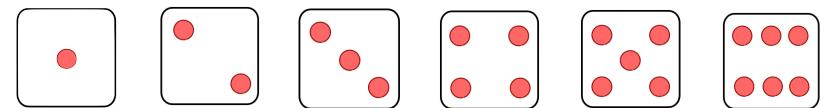
2 states

fair dice  
biased dice

starting with the fair dice

A	<table border="1"><tr><td>0,9</td><td>0,1</td></tr><tr><td>0,1</td><td>0,9</td></tr></table>	0,9	0,1	0,1	0,9
0,9	0,1				
0,1	0,9				

E	<table border="1"><tr><td>1/6</td><td>1/6</td><td>1/6</td><td>1/6</td><td>1/6</td><td>1/6</td></tr><tr><td>3/13</td><td>2/13</td><td>2/13</td><td>2/13</td><td>2/13</td><td>2/13</td></tr></table>	1/6	1/6	1/6	1/6	1/6	1/6	3/13	2/13	2/13	2/13	2/13	2/13
1/6	1/6	1/6	1/6	1/6	1/6								
3/13	2/13	2/13	2/13	2/13	2/13								

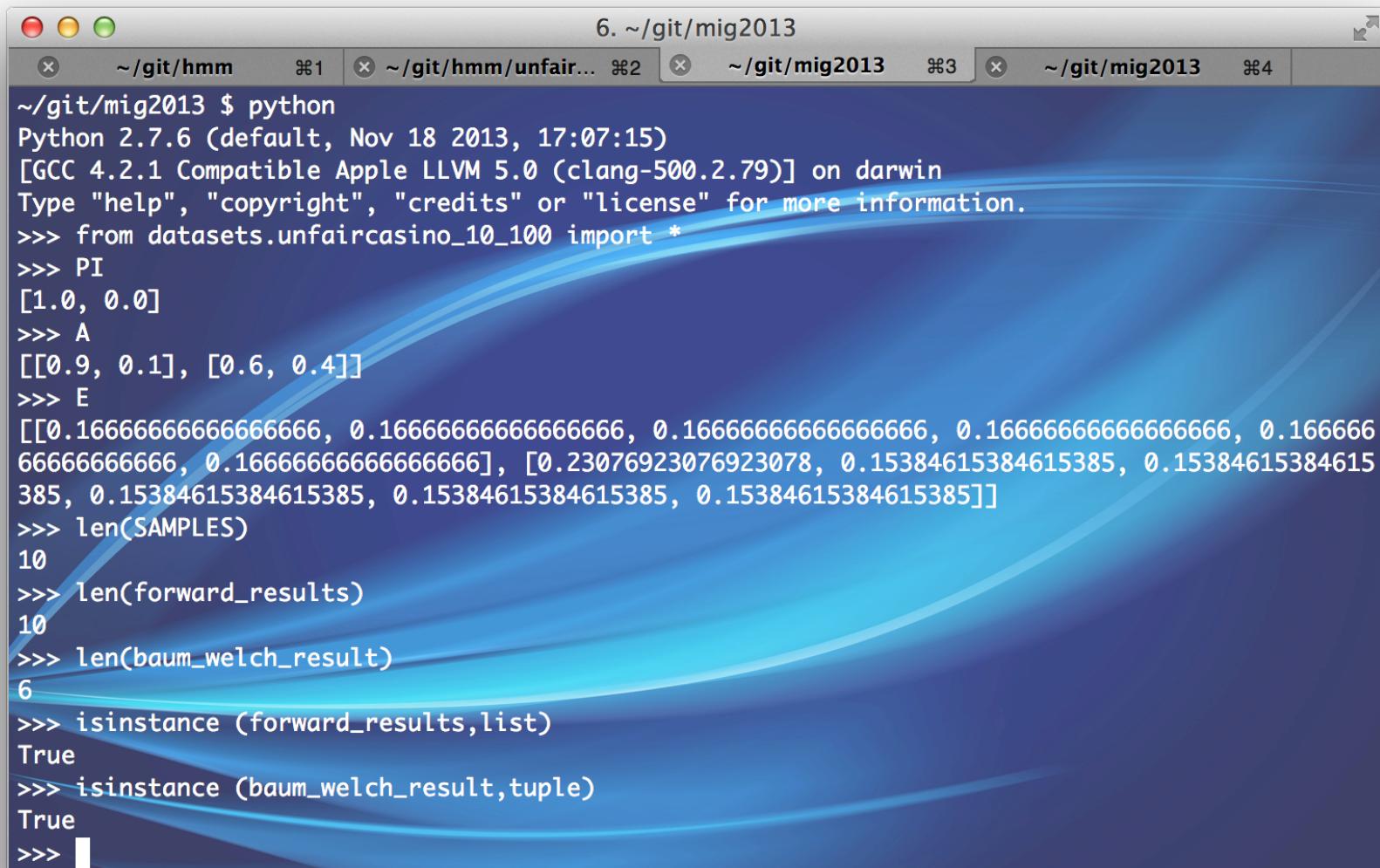


$\Pi$	<table border="1"><tr><td>1</td><td>0</td></tr></table>	1	0
1	0		

# Available datasets

---

- For comparing your results: see datasets/



The screenshot shows a Mac OS X terminal window titled "6. ~/git/mig2013". The window has four tabs at the top: "~git/hmm" (⌘1), "~git/hmm/unfair..." (⌘2), "~git/mig2013" (selected tab, ⌘3), and "~git/mig2013" (⌘4). The terminal content is as follows:

```
~/git/mig2013 $ python
Python 2.7.6 (default, Nov 18 2013, 17:07:15)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.2.79)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from datasets.unfaircasino_10_100 import *
>>> PI
[1.0, 0.0]
>>> A
[[0.9, 0.1], [0.6, 0.4]]
>>> E
[[0.1666666666666666, 0.1666666666666666, 0.1666666666666666, 0.1666666666666666, 0.1666666666666666, 0.1666666666666666, 0.1666666666666666], [0.23076923076923078, 0.15384615384615385, 0.15384615384615385, 0.15384615384615385, 0.15384615384615385, 0.15384615384615385]]
>>> len(SAMPLES)
10
>>> len(forward_results)
10
>>> len(baum_welch_result)
6
>>> isinstance (forward_results,list)
True
>>> isinstance (baum_welch_result,tuple)
True
>>> 
```

# Viterbi – decoding

---

- Given an observation sequence  
find out most likely path that has caused this sequence  
a.k.a. **Viterbi path**
- **delta[i][t]**  
best probability of path **until t** that **ends up at i**  
(and of course emits sequence until t)
- **psi[i][t]**  
state at t-1 for the path that corresponds to *delta*  
 $\psi[i][0]$  undefined / does not matter

# Viterbi – details

---

- Initialization

$$\delta[i][0] = \Pi[i] * E[i][\text{sequence}[0]]$$

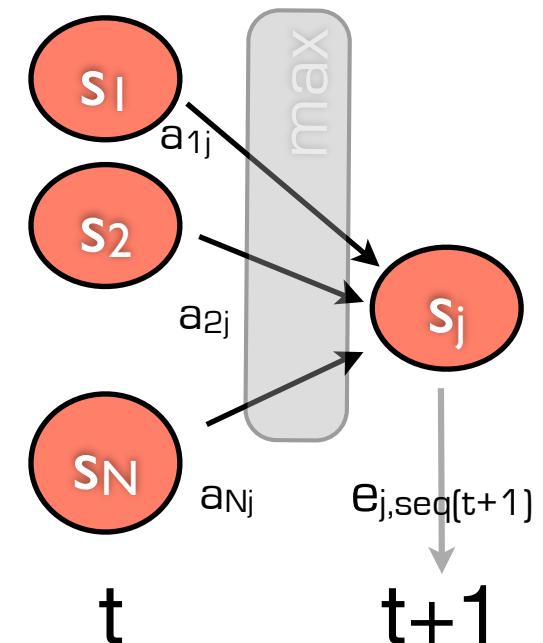
$$\psi[i][0] = \text{None}$$

- Recurrence

$$\delta[j][t+1] = \backslash$$

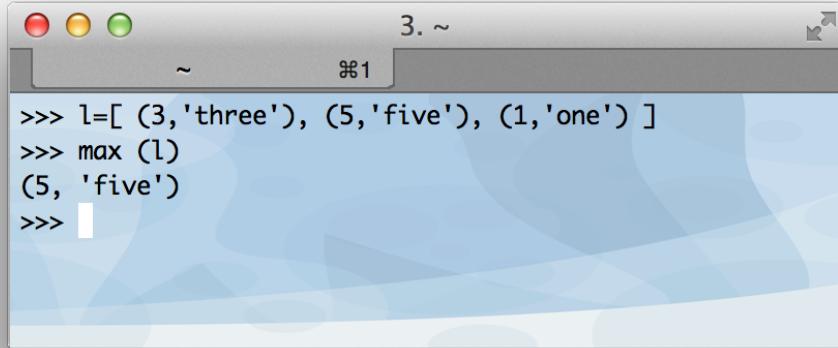
max ( [  $\delta[i][t] * A[i][j]$  ] for  $i$  in range( $N$ ) ) \*  $E[j][\text{seq}[t+1]]$

$$\psi[j][t] = \text{argmax} ( \text{same expression} )$$



# Viterbi – a useful trick

---



A screenshot of a Mac OS X terminal window. The title bar says "3. ~" and the tab bar shows "⌘1". The terminal window contains the following Python code:

```
>>> l=[(3,'three'), (5,'five'), (1,'one')]  
>>> max(l)  
(5, 'five')  
>>> |
```

$(\text{proba}, \text{rank}) = \max \left( [ (\delta[i][t-1] * A[i][j], i) \text{ for } i \in \text{range}(N) ] \right)$

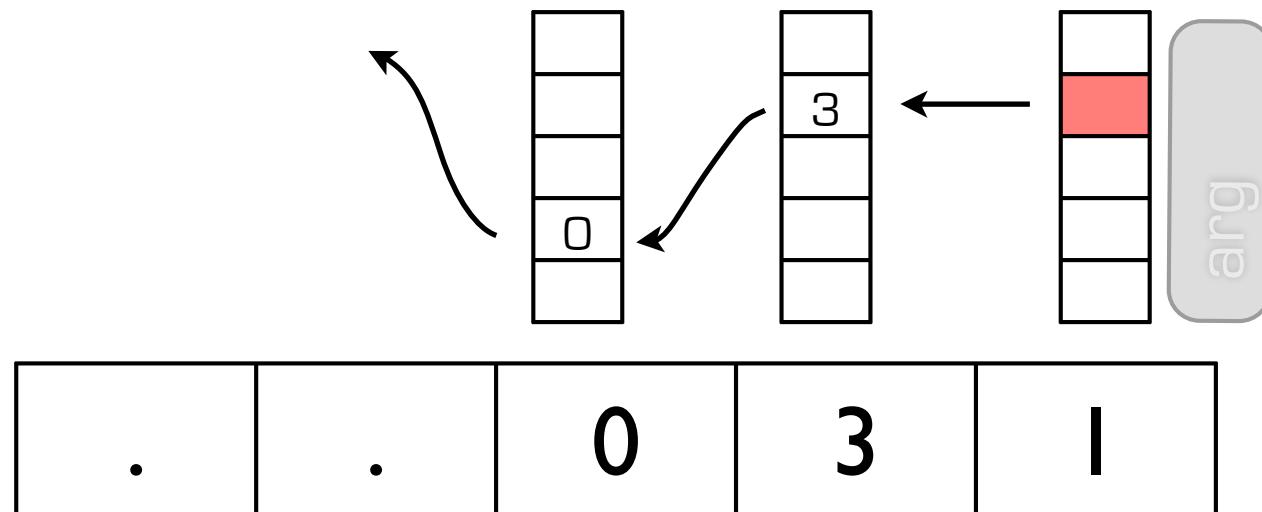
$\delta[j][t] = \text{proba} * E[j][\text{seq}[t]]$

$\psi[j][t] = \text{rank}$

# Viterbi - finalization

- retrieve full path

$\psi[T-2] \ \psi[T-1] \ \delta[T-1]$



- Viterbi is a greedy algorithm too

- $\Theta(N^2 * T)$  in space and time

# One possible implementation for forward

---

```
# iterative style
def forward (Pi, A, E, sequence):
    N = len(A)
    T = len(sequence)
    states = range(N)

    alpha = [ [ 0. for t in range(T) ] for i in states ]

    # time=0
    for i in states: alpha[i][0]=Pi[i] * E[i][sequence[0]]

    # fill in from time=1
    for t in range (1,T):
        for j in states:
            for i in states:
                alpha[j][t] += alpha[i][t-1]*A[i][j]
            # multiply once rather than for each iteration
            alpha[j][t] *= E[j][sequence[t]]
    # resulting proba is a sum from that last time column
    total = 0.
    for i in states: total += alpha[i][T-1]
    return (total, alpha)
```

# One possible implementation for backward

---

```
from operator import add
# return ( proba, beta ) -- beta dimension: beta[i][t]
# functional style
def backward (Pi, A, E, sequence):
    N = len(A)
    T = len(sequence)
    states = range(N)

    beta = [ [ 0. for t in range(T) ] for i in states ]

    # beta starts with 1's on last time
    for i in states: beta[i][T-1]=1.

    # starting with time but last, i.e. T-2
    for t in range (T-2, -1, -1):
        for i in range (N):
            beta[i][t] =
                reduce (add, [ beta[j][t+1] * A[i][j] * E[j][sequence[t+1]] for j in states ] )
    # resulting proba, summing on time=0
    total = reduce (add, [ Pi[i]*E[i][sequence[0]]*beta[i][0] for i in states ] )
    return (total,beta)
```

# Implementation

---

- implement a function that can be used like this  
 $(\text{proba}, \text{path}, \text{delta}, \text{psi}) = \text{viterbi}(\text{Pi}, \text{A}, \text{E}, \text{sequence})$
- (although delta and psi are not really useful outside)

# Baum-Welch – learning

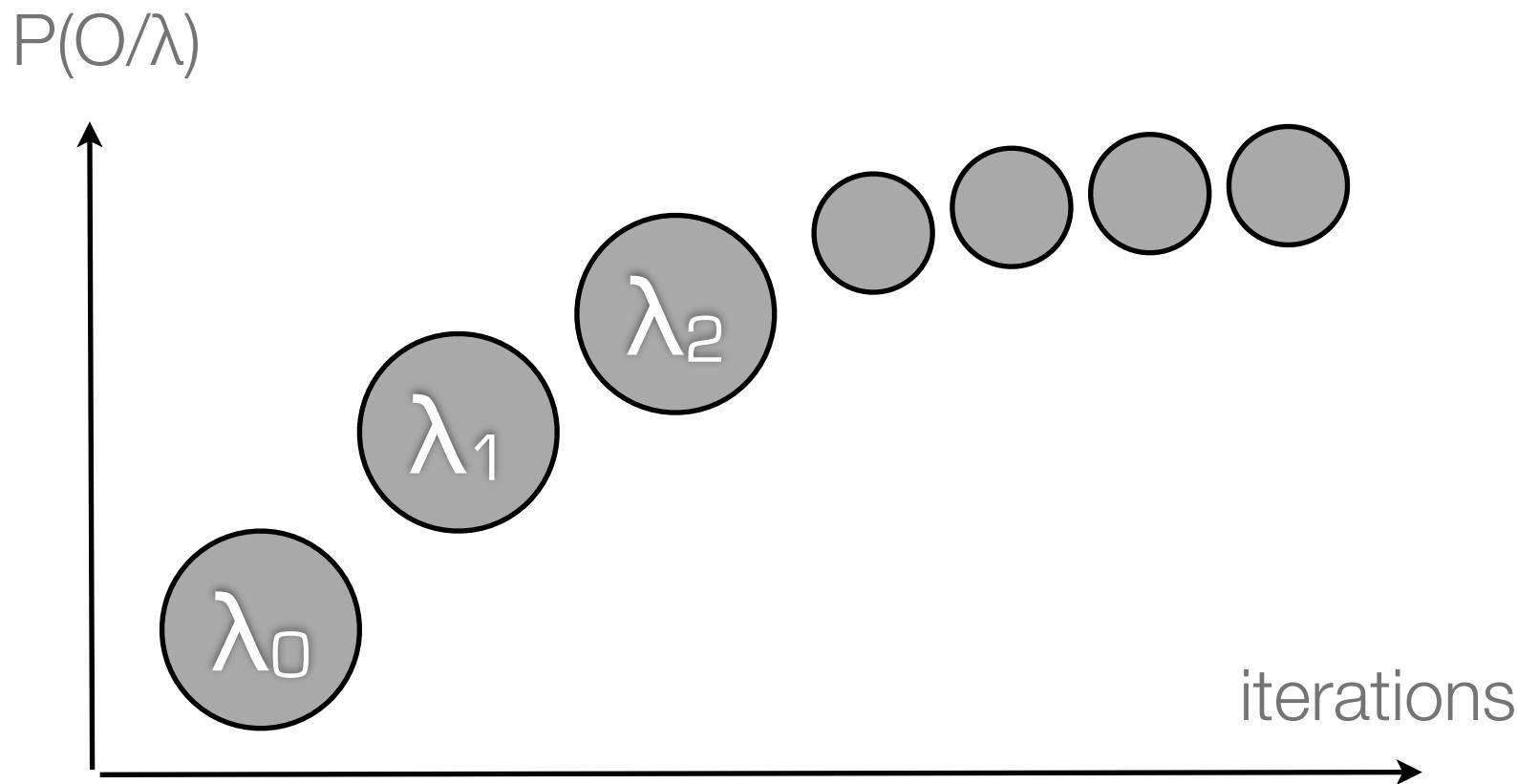
---

- Problem: given a set of output sequences  
Find out the “best” HMM that would give this out
- Improve model  $\lambda = (A, E, \text{PI})$  by successive iterations
- maximize likelihood over  $\lambda$   
likelihood defined as the **product** of probabilities of all sequences in a sample
- in our case, multiply  $P(O/\lambda)$  over sequences

# Expectation Maximization (EM)

---

- Baum-Welch is a special case of “Expectation - Maximization” class of algorithms



# Baum Welch basics

---

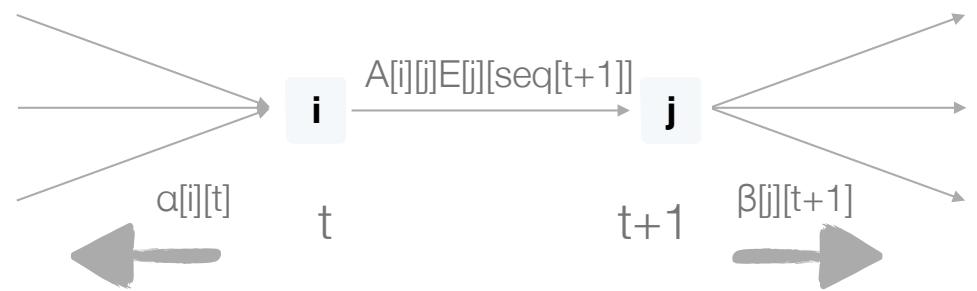
- Given sequence seq, and model  $\lambda$
- $\boldsymbol{\xi}_t[i, j]$   
probability to be in state **i** at time **t**, and in state **j** at **t+1**
- $\boldsymbol{\gamma}_t[i]$   
probability to be in state **i** at time **t**

# computing xi and gamma

---

- $\xi_t(i, j)$  probability to be in **i** at **t**, **j** at **t+1**(knowing seq)

$$\xi_t(i, j) = \frac{\alpha_t(i) A_{i,j} E_{j,seq_{t+1}} \beta_{t+1}(j)}{P(seq/\lambda)}$$



- $\gamma_t(i)$  probability to be in **i** at **t** (knowing seq)

$$\gamma_t(i) = \sum_{j=0}^{N-1} \xi_t(i)(j)$$

# Rationale for re-estimation

---

$\sum_{t=0}^{T-2} \gamma_t(i) = \text{expected number of transitions from state } i$

$\sum_{t=0}^{T-2} \xi_t(i,j) = \text{expected number of transitions from state } i \text{ to } j$

- note that one instant is not taken into account  
we are counting transitions, so there are T-1  
will be used for re-estimating PI instead

# Re-estimation

---

$$\bar{\pi}_i = \gamma_0(i)$$

$$\bar{A}_{ij} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions from } i}$$

$$= \frac{\sum_{t=0}^{T-2} \xi_t(i, j)}{\sum_{t=0}^{T-2} \gamma_t(i)}$$

$$\bar{E}_{jo} = \frac{\text{expected number of times in } j \text{ and observing } o}{\text{expected number of times in } j}$$

$$= \frac{\sum_{t=0}^{T-1} \gamma_t(j)}{\sum_{t=0}^{T-1} \gamma_t(j)}$$

*so that  $o = \text{seq}(t)$*

reminder from previous slides

\* in **i** at **t** and in **j** at **t+1**

$$\xi_t(i, j) = \frac{\alpha_t(i) A_{i,j} E_{j,\text{seq}_{t+1}} \beta_{t+1}(j)}{P(\text{seq}/\lambda)}$$

\* in **i** at **t**

$$\gamma_t(i) = \sum_{j=0}^{N-1} \xi_t(i)(j)$$

# Convergence criteria

---

- monitor overall  $P(\text{seq})$  at each iteration
- should grow asymptotically
- stop when  $(P'/P) < (1+\varepsilon)$
- $\varepsilon$  passed as argument

# Deal with multiple sample sequences

$$\bar{\pi}_i = \gamma_0(i)$$

Average on sequences

$$\bar{A}_{ij} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions from } i}$$

Sum over sequences

$$= \frac{\sum_{t=0}^{T-2} \xi_t(i, j)}{\sum_{t=0}^{T-1} \gamma_t(i)}$$

Sum over sequences

$$\bar{E}_{jo} = \frac{\text{expected number of times in } j \text{ and observing } o}{\text{expected number of times in } j}$$

Sum over sequences

$$= \frac{\sum_{t=0}^{T-1} \gamma_t(j) \delta(o, \text{seq}(t))}{\sum_{t=0}^{T-1} \gamma_t(j)}$$

Sum over sequences

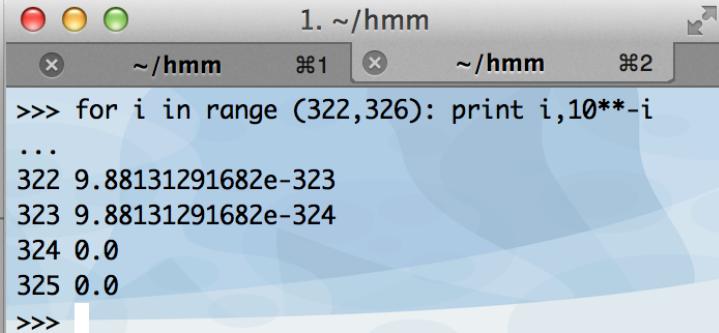
# Implementation

---

- implement a function that can be used like this

```
(PI', A', E', proba, iteration,time) = \  
baum_welch (PI, A, E,sequenceS,\  
max_iters=100, convergence=1.e-4)
```

# Scaling



```
1. ~/hmm
~ /hmm ~ /hmm ~ /hmm ~ /hmm
>>> for i in range (322,326): print i,10**-i
...
322 9.88131291682e-323
323 9.88131291682e-324
324 0.0
325 0.0
>>>
```

- If  $P_{\max}$  is the maximum of the terms in A and E
- Order of magnitude for e.g.  $\alpha[i][t]$  :  $P_{\max}^{2T}$
- Even if  $P_{\max}$  is, say, 0.9,  $T=300$  gives you fairly small values
- Quickly reach limits of hardware implementation
- Solution: for each time t
  - select some relevant factor (e.g. based on sum of values)
  - store in memory real value multiplied by factor

# Continuous Hidden Markov Models

# Continuous Hidden Markov Model

---

- Replace finite set of  $O$  signals (outputs)
- With a finite set of  $O$  gaussian distributions in  $R^d$
- Each of these being defined by
  - an average  $\mu = \{\mu_1, \mu_2, \dots \mu_d\}$
  - a covariance matrix  $\sigma_{ij}$  in dimension  $d$
- Then each state is attached a linear stochastic combination of these gaussians (a **mixture**)

# Continuous HMM – example

- 3 Outputs:

$$G1 : \mu=(4., 8.), \quad \sigma=(1., 0.25)$$

$$G2 : \mu=(-4., -8.), \quad \sigma=(1., 4.)$$

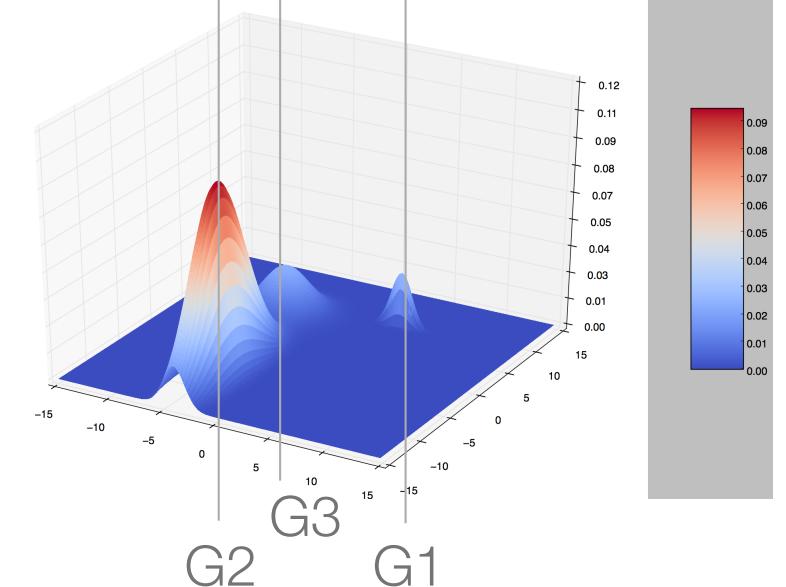
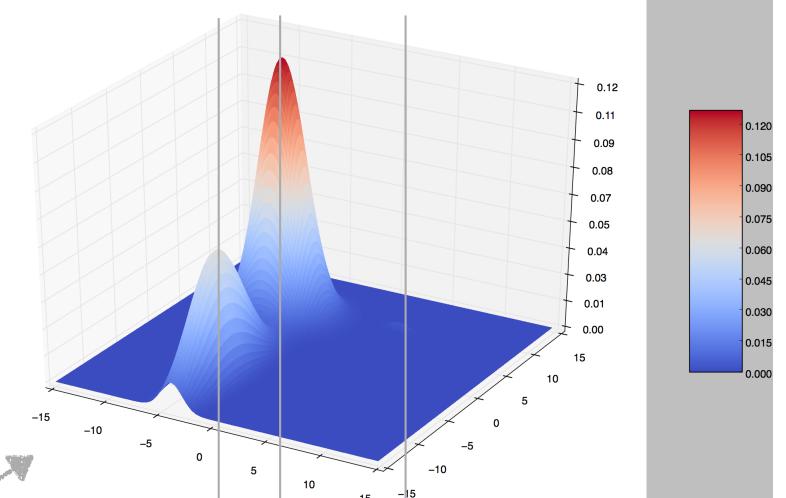
$$G3 : \mu=(-6., 6.), \quad \sigma=(2., 2)$$

say  $G1 \sim$ tired,  
 $G2 \sim$ feverish,  $G3 \sim$  feels good

- 2 states:

$$E[1]: (.1, .1, .8)$$

$$E[2]: (.7, .15, .15)$$



# Gaussian distribution - single dimension

- a.k.a. Normal Distribution (wiki)

- basic shape

$$f(x) = e^{-x^2}$$

- normalize

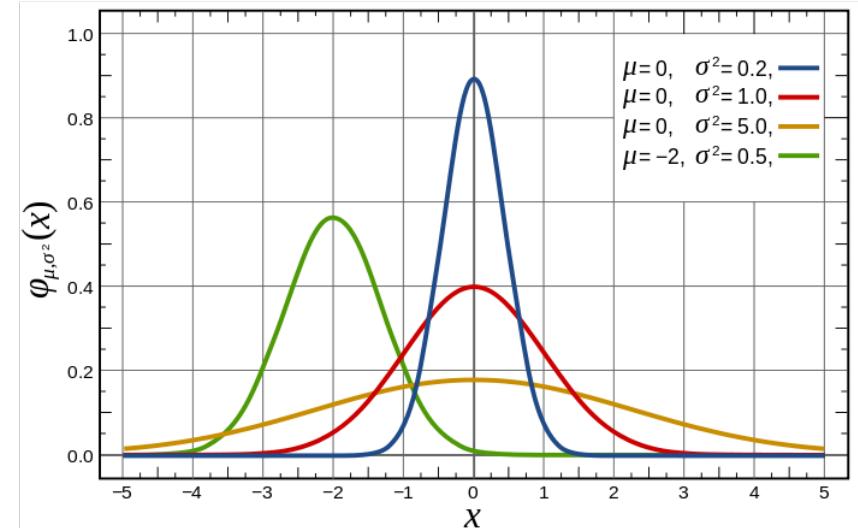
$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

- translate

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-\mu)^2}$$

- scale

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$



# General form of d-dimension gaussian

---

- a.k.a. multivariate normal distribution (wiki)
- $\mu$  average;  $\Sigma$  covariance  
 $\Sigma$  is positive semidefinite and symmetric

$$\mu \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d,d}, G_{\mu,\Sigma} : \mathbb{R}^d \mapsto \mathbb{R}$$

$$G_{\mu,\Sigma}(X) = \frac{1}{\sqrt{(2\pi)^d \cdot \det(\Sigma)}} e^{(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu))}$$

- check out “covariance matrix” in wiki

# Degenerate case

---

- Diagonal covariance matrix (unrelated data)

$$\Sigma = \begin{pmatrix} \sigma_1^2 & 0 & \cdot & \cdot & 0 \\ 0 & \sigma_2^2 & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \sigma_{d-1}^2 & 0 \\ 0 & \cdot & \cdot & 0 & \sigma_d^2 \end{pmatrix}$$

$$G_{\mu, \Sigma}(X) = \frac{1}{\prod_{i=1}^d \sigma_i \sqrt{(2\pi)^d}} e^{(-\frac{1}{2} \sum_{i=1}^d (\frac{x_i - \mu_i}{\sigma_i})^2)}$$

# Possible adaptation from discrete HMMs

---

- Replace “ $E[i][seq[t]]$ ”
- With the evaluation of the gaussian mixture defined for  $i$ ,  
on that point in space  $seq[t]$

# Source Code Management systems

# Source Code Management systems

---

- purpose
  - collaborative work
  - keep track of changes
  - changes are done concurrently
  - teams do not necessarily know of each other
- git
  - <http://git-scm.com>
  - decentralized
  - used by many open source projects
- svn
  - <http://subversion.tigris.org>
  - deprecated / previous generation
  - simpler mental model, but centralized
- mercurial
  - <http://mercurial.selenic.com> is nice too

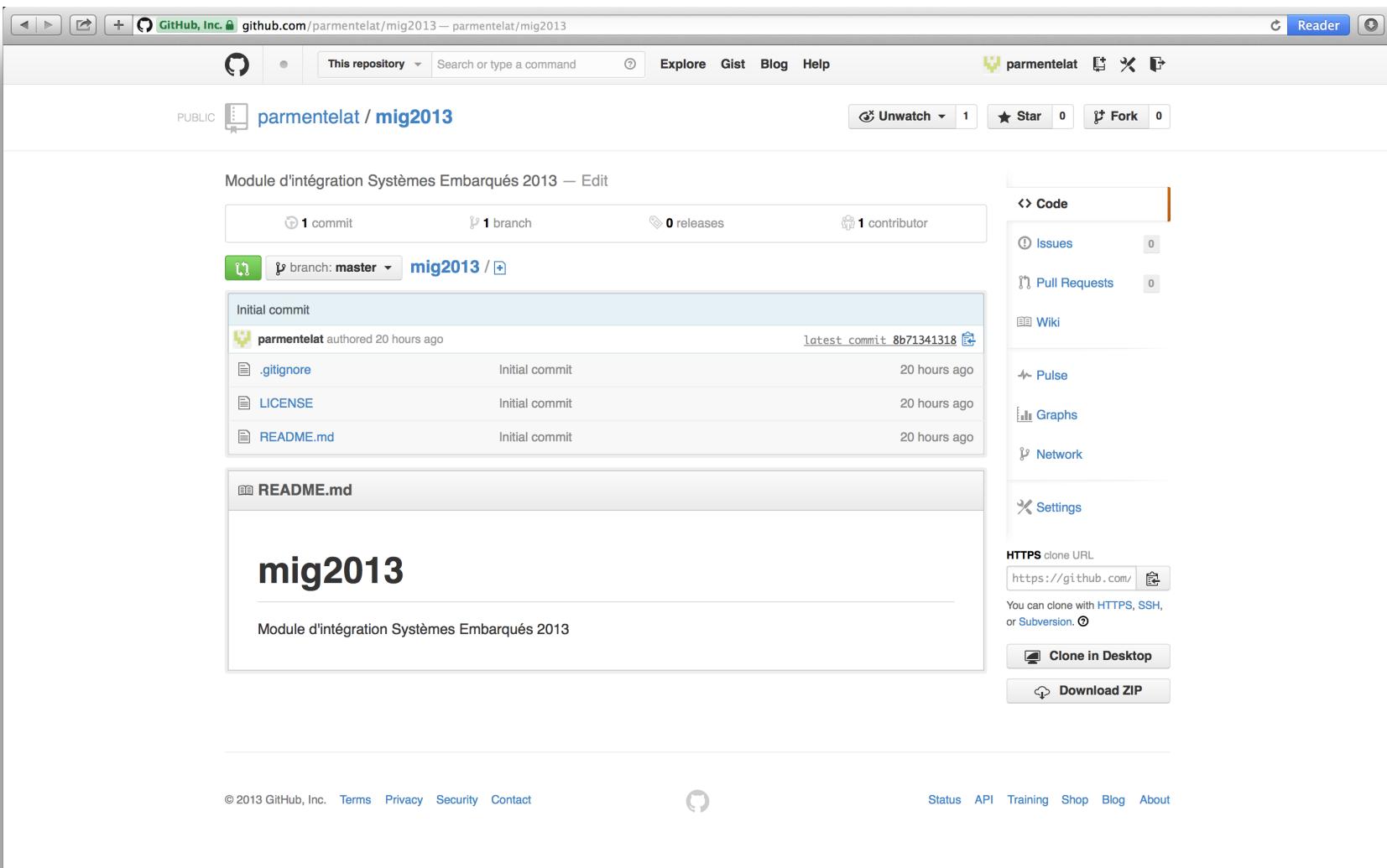
# git basics

---

- git init initialize git from a working directory
- git clone create a local working dir from remote
- git add put changes aside for next commit
- git commit create a commit
- git pull get changes from remote
- git push push your commits on remote
- git stash hide local changes under carpet

Check out UI tools : <http://www.sourcetreeapp.com>

# example (1) - create repo in github



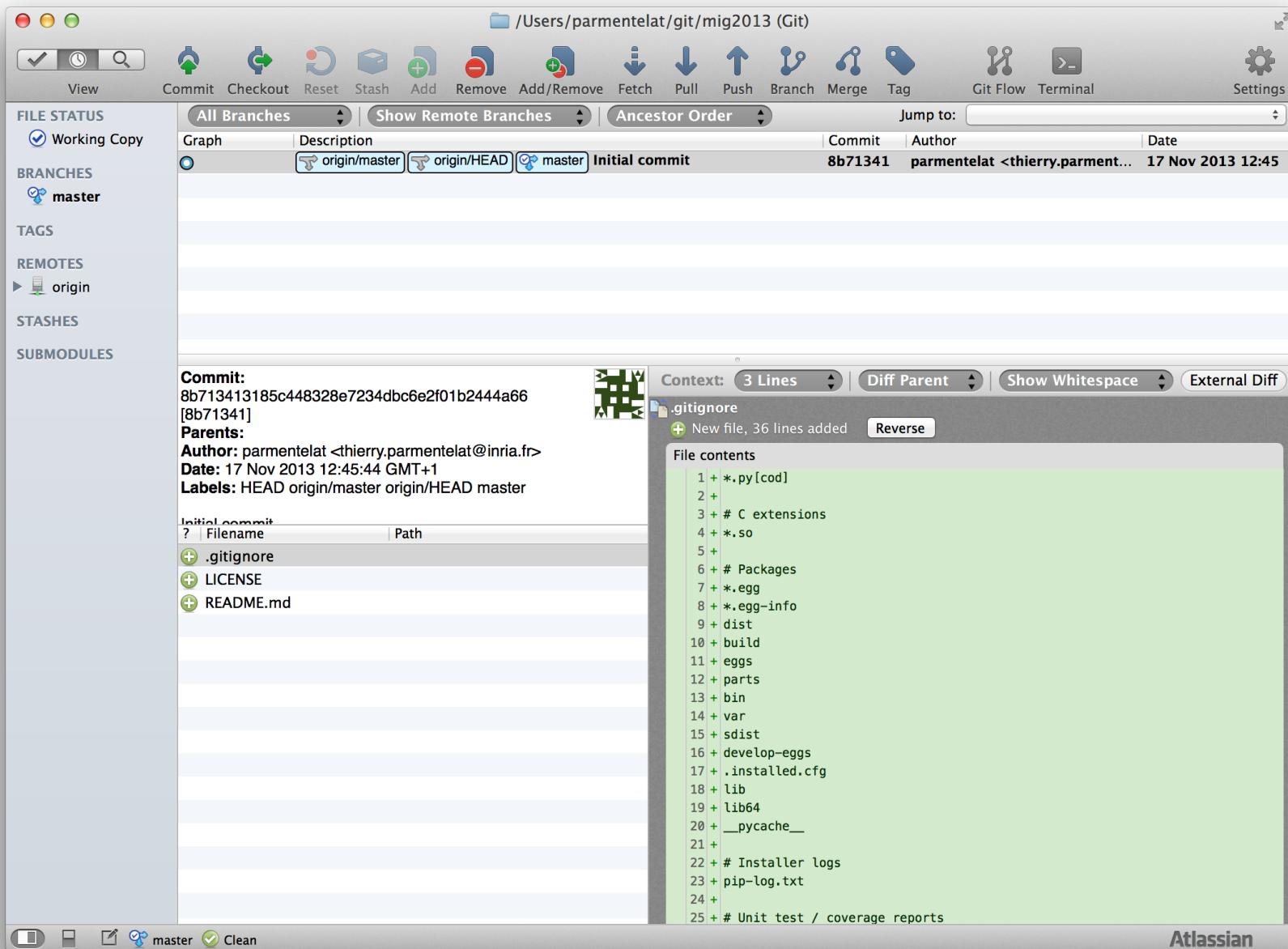
## example (2) - clone repo locally

---

The image shows a terminal window with a blue wavy background. The title bar says "3. ~/git/mig2013". The window title is "git mig2013". The command entered was "git clone https://github.com/parmentelat/mig2013.git". The output shows the cloning process: counting objects (5), compressing objects (100% 4/4), total (5), reused (0), and unpacking objects (100% 5/5). It also shows the user navigating to the directory and listing the contents, which include "LICENSE" and "README.md".

```
~/git $ git clone https://github.com/parmentelat/mig2013.git
Cloning into 'mig2013'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
Checking connectivity... done
~/git $ cd mig2013/
~/git/mig2013 $ ls
LICENSE      README.md
~/git/mig2013 $
```

# example (3) - visualize local repo with sourcetree



## example (4) - two working directories

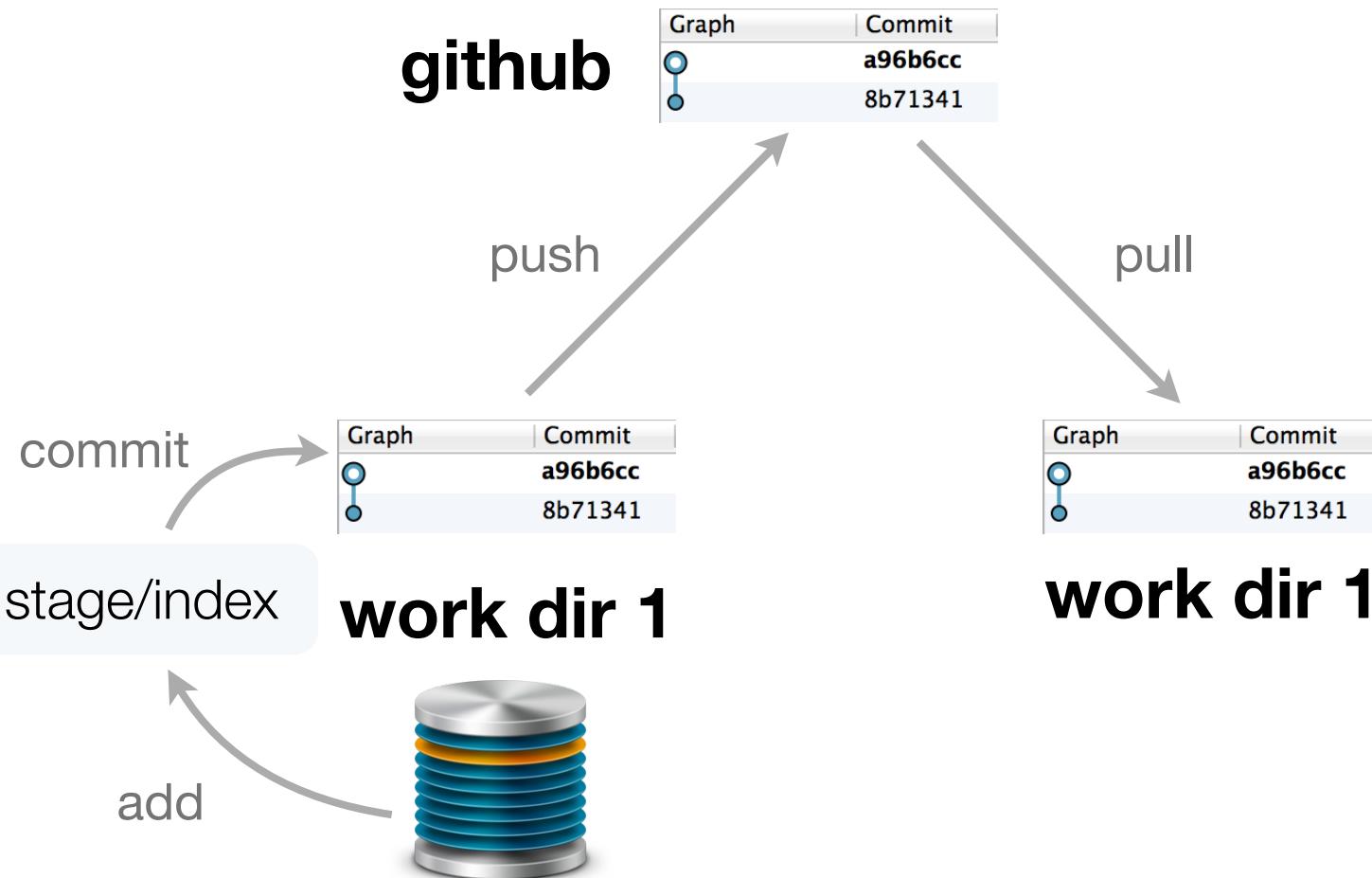
---

```
$ git ls-files playground.txt  
$ git add playground.txt  
$ git commit -m "my message"  
$ git ls-files playground.txt  
playground.txt  
$ git push
```

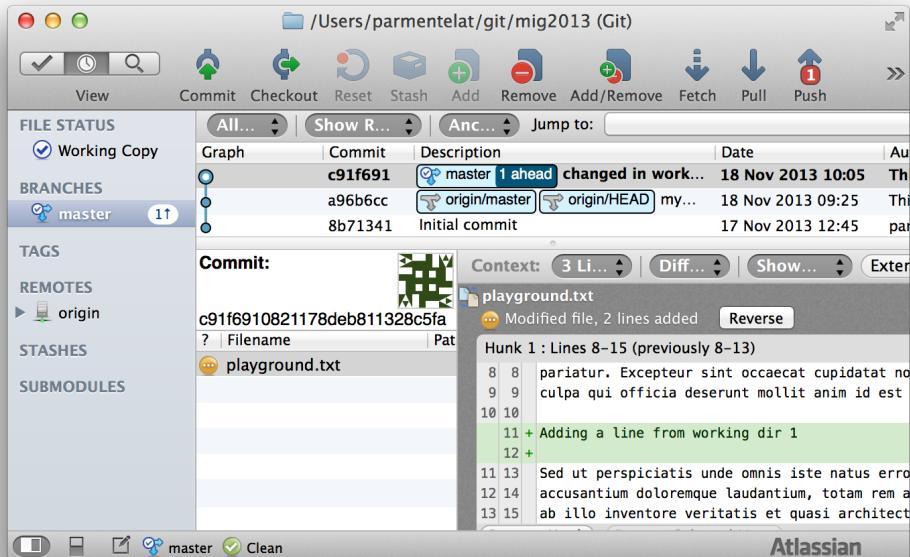
```
$ git ls-files playground.txt  
$  
$  
$ git ls-files playground.txt  
$  
$  
$ git pull  
$ git ls-files playground.txt  
$ playground.txt
```

# example (5) - the repositories at work

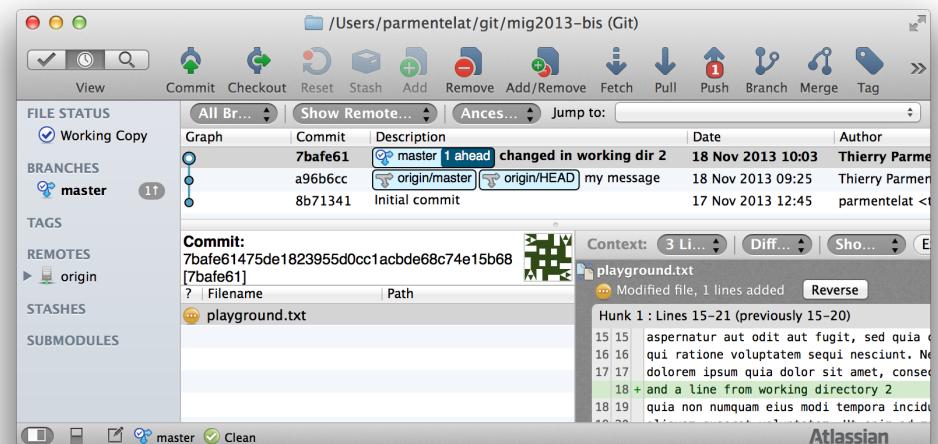
---



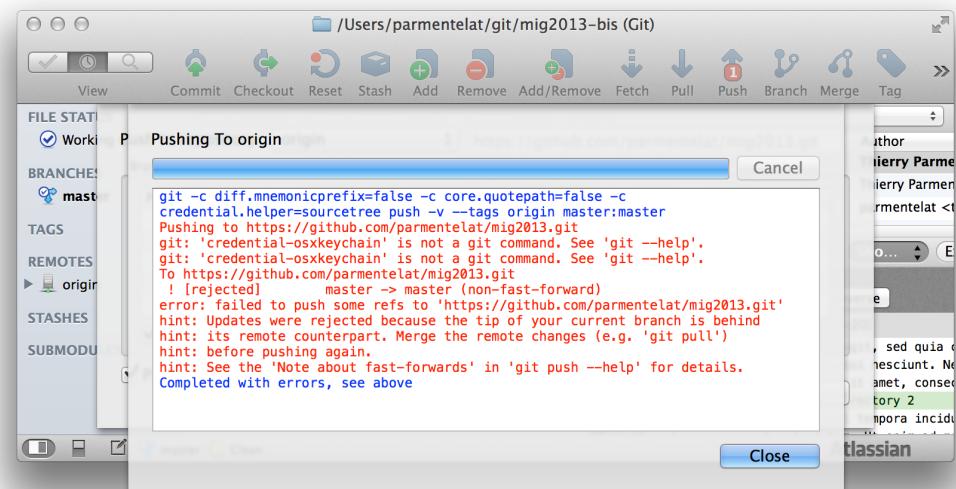
# example (6) - merging two concurrent changes



user 1 can push fine

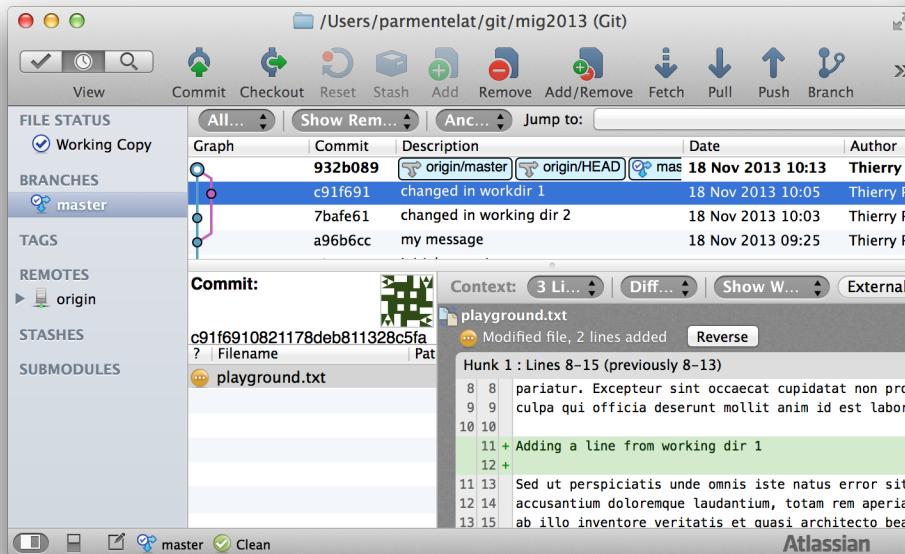


user 2 gets conflicts when pushing

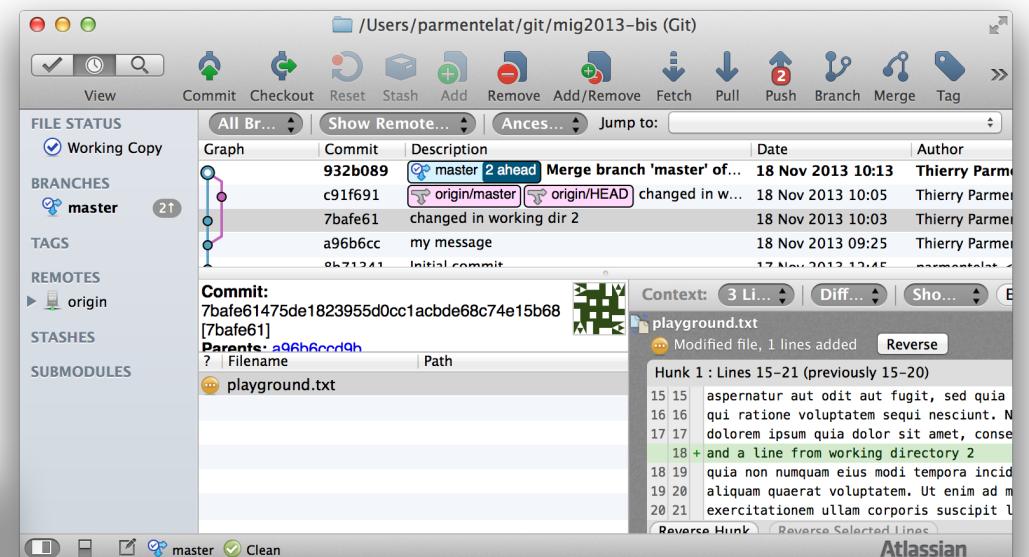


# example (7) - merging - continued

user 1 sees once he pulls again



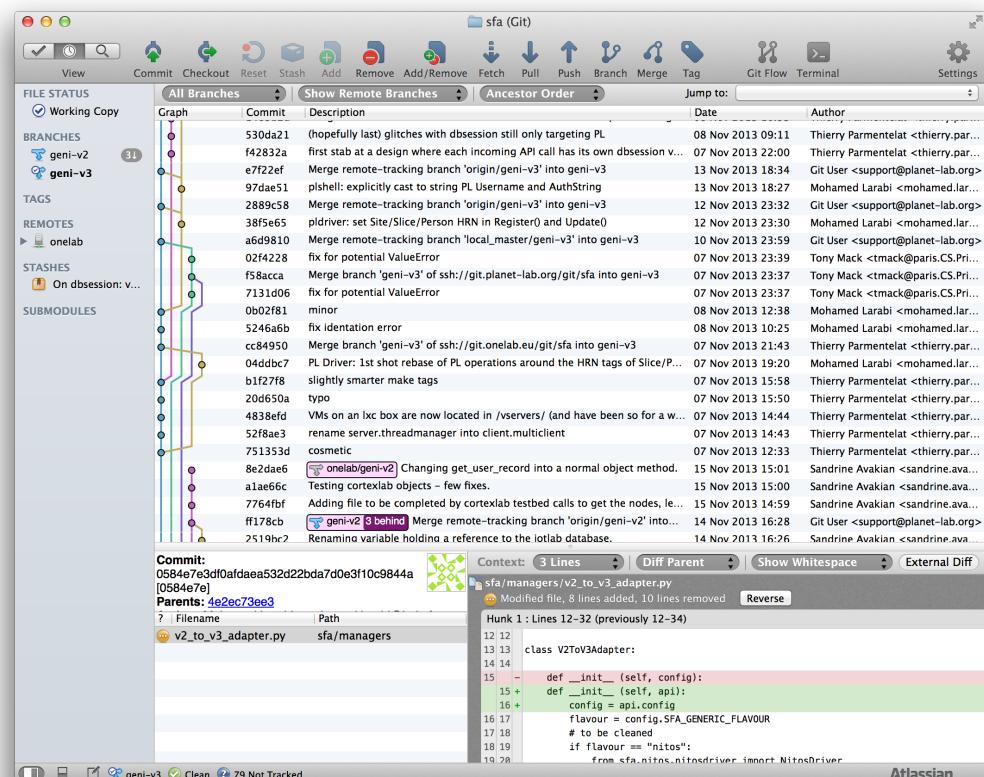
user 2 needs to pull first



user 2 can then push fine

# Good practices

- Branches are easy to create, merge, delete..
- Use as many branches as needed
- Typically one per “feature”



The screenshot shows the Atlassian SourceTree application interface for a Git repository named 'sfa'. The main window displays a detailed commit history. On the left, there are sections for 'FILE STATUS', 'BRANCHES' (showing 'geni-v2' and 'geni-v3'), 'TAGS', 'REMOTES' (showing 'onelab'), and 'STASHES'. The central area features a 'Graph' view showing the commit history, which is highly branched, indicating frequent feature development and rebase operations. Below the graph, a table lists individual commits with columns for 'Commit', 'Description', 'Date', and 'Author'. A specific commit, '0584e7e3df0afdaea532d22bda7d0e3f10c9844a [0584e7e]', is selected, showing its details in the bottom panel. This commit is a merge from 'geni-v2' into 'geni-v3'. The commit message is: 'Merge remote-tracking branch 'origin/geni-v2' into geni-v3'. The commit date is 07 Nov 2013 12:20, and the author is 'Thierry Parmentelat <thierry.par...>'. The commit details show code changes in the file 'v2\_to\_v3\_adapter.py' under the 'sfa/managers' directory, specifically adding code for 'V2ToV3Adapter' and setting 'flavour' to 'SFA\_GENERIC\_FLAVOUR'.

Date	Author
08 Nov 2013 09:11	Thierry Parmentelat <thierry.par...
07 Nov 2013 22:00	Thierry Parmentelat <thierry.par...
13 Nov 2013 18:34	Git User <support@planet-lab.org>
13 Nov 2013 18:27	Mohamed Larabi <mohamed.lar...
12 Nov 2013 23:32	Git User <support@planet-lab.org>
12 Nov 2013 23:30	Mohamed Larabi <mohamed.lar...
07 Nov 2013 23:39	Tony Mack <tmac@paris.cs.pr...
10 Nov 2013 23:59	Git User <support@planet-lab.org>
07 Nov 2013 23:37	Tony Mack <tmac@paris.cs.pr...
07 Nov 2013 23:37	Tony Mack <tmac@paris.cs.pr...
08 Nov 2013 12:38	Mohamed Larabi <mohamed.lar...
08 Nov 2013 10:25	Mohamed Larabi <mohamed.lar...
07 Nov 2013 21:43	Thierry Parmentelat <thierry.par...
07 Nov 2013 19:20	Mohamed Larabi <mohamed.lar...
07 Nov 2013 15:58	Thierry Parmentelat <thierry.par...
07 Nov 2013 15:50	Thierry Parmentelat <thierry.par...
07 Nov 2013 14:44	Thierry Parmentelat <thierry.par...
07 Nov 2013 14:43	Thierry Parmentelat <thierry.par...
07 Nov 2013 12:33	Thierry Parmentelat <thierry.par...
15 Nov 2013 15:01	Sandrine Avakian <sandrine.av...
15 Nov 2013 15:00	Sandrine Avakian <sandrine.av...
15 Nov 2013 14:59	Sandrine Avakian <sandrine.av...
14 Nov 2013 16:28	Git User <support@planet-lab.org>
14 Nov 2013 16:26	Sandrine Avakian <sandrine.av...

ssh basics

# ssh history and purpose

---

- successor of (very unsecure) *rsh* (remote shell)
- ssh = secure shell
- basic purpose : remote terminal
- advanced purpose : all sorts of secure tunnels / bouncing
- esp. e.g. rsync (to keep files in sync), git, ...

# ssh authentication mechanism(s)

---

- **password** authentication is **evil**, don't ever enable
- use **public key** authentication only
- ssh-keygen : create a key pair
- keep private key (id\_rsa), well... private
  - never out of your computer
  - watch out access rights
  - private keys are password-protected
- expose public key (id\_rsa.pub) to your peers

# ssh public key authentication - basics

---

- how to enable access
  - add public key in `~/.ssh/authorized_keys`
  - again watch out for access rights
- how it works
  - $E_{\text{Public}} \circ E_{\text{private}} = E_{\text{private}} \circ E_{\text{public}} = \text{Identity}$
  - challenge remote to “be” X (to have private key X)
  - send  $E_{\text{public}}(\text{message})$  over the wire
  - $E_{\text{private}}$  is required to decode message