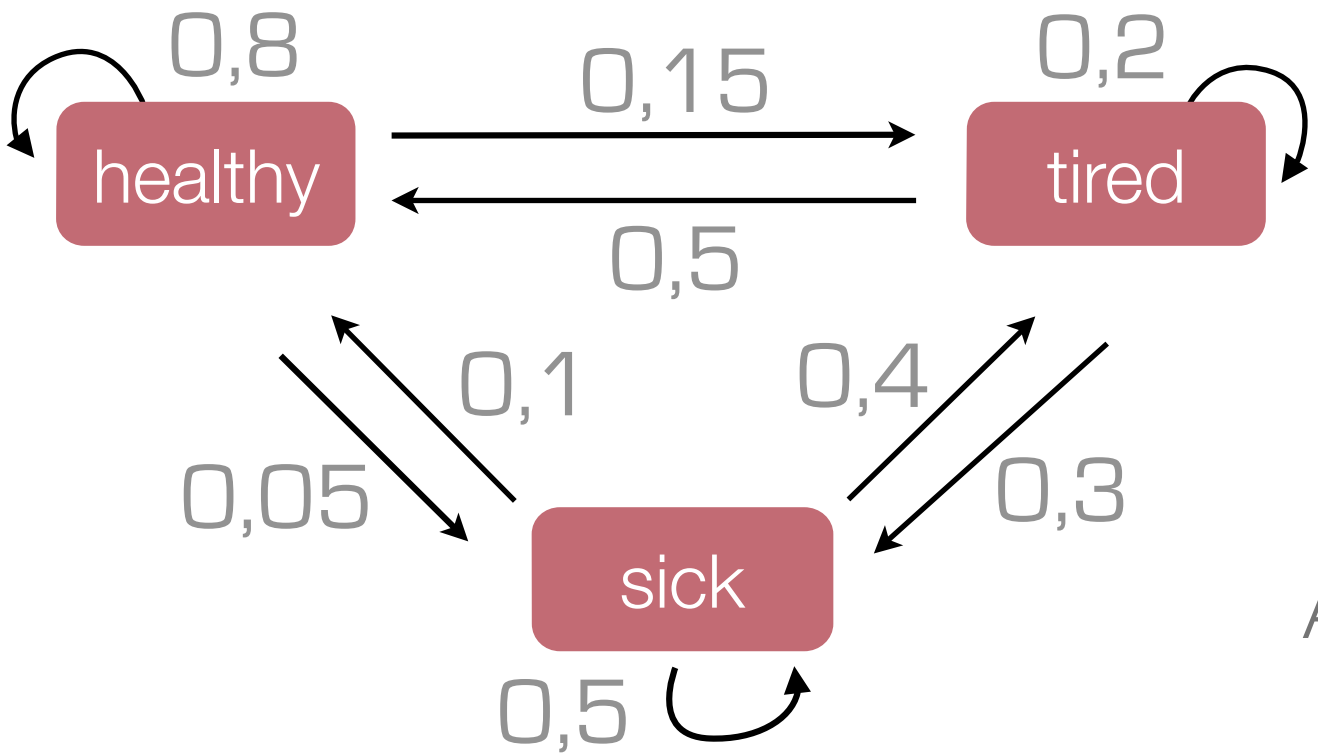
Hidden Markov Models

Thierry Parmentelat — INRIA
thierry.parmentelat@inria.fr

Markov Models



- model : λ=(A,π)
- A: Transitions
- π: p(initial state=i)

	0,8	0,15	0,05	Σ=1
A	0,5	0,2	0,3	
	0,1	0,4	0,5	

π	0,7	0,2	0,1

Markov Models

- Markov hypothesis
  - behaviour depends only on current state (**not** on history)
- Observation
  - $E_1E_2..E_t..$: 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 $E_1 E_2 .. E_t$

    $$proba = \pi(E_1) * \prod_{i=2..t} A_{(Ei-1)(Ei)}$$

  - CS: sequence seq[t]

    $$proba = pi\ [seq[0]] * \prod_{i=2..t} (\ A\ [seq[i-1]]\ [seq[i]]\ )$$

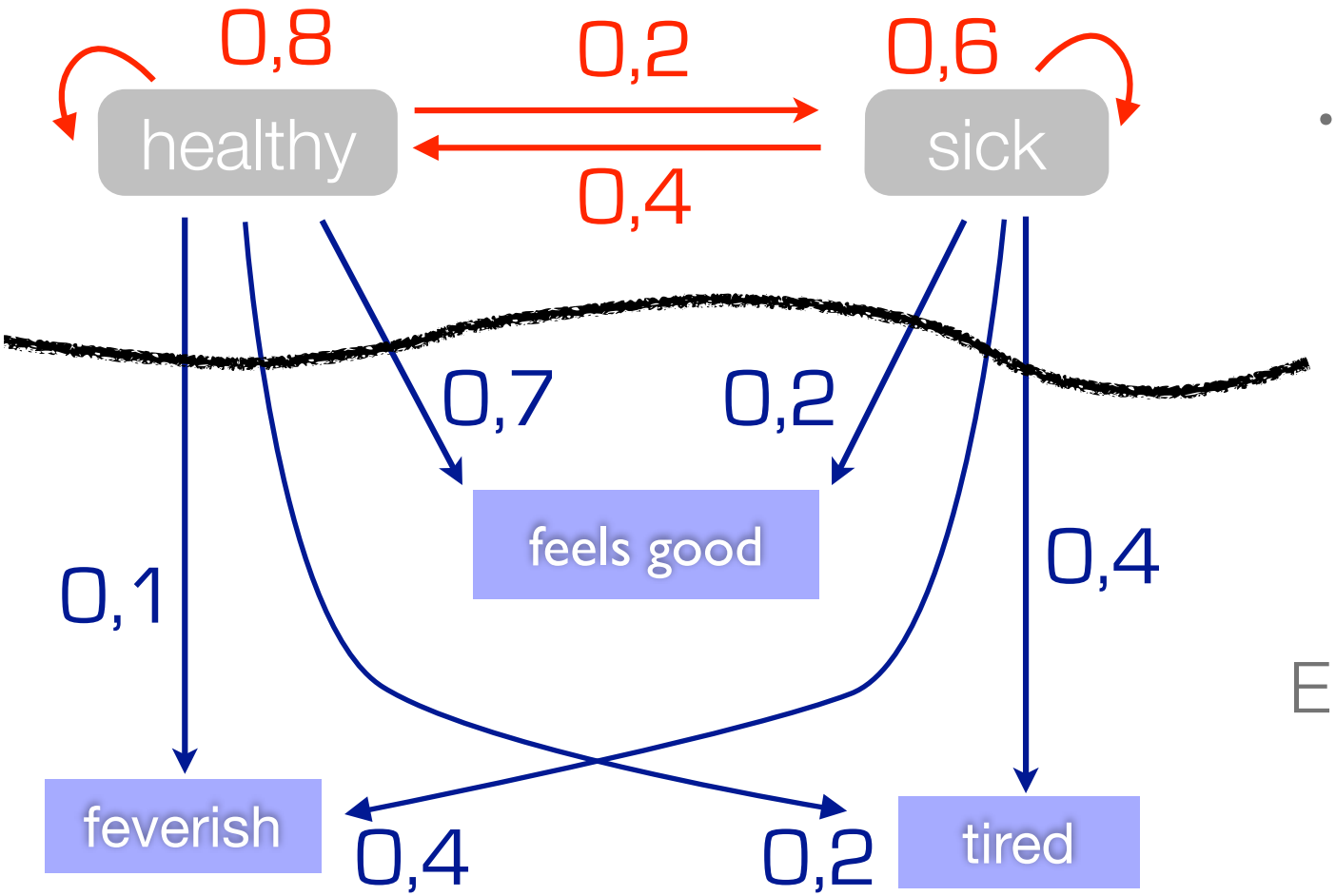
From now on,
we will have
all indices start at 0

Python features, and modules of interest

- range — range(5) -> [0, 1, 2, 3, 4] — range(1,5) -> [1, 2, 3, 4]
- [function(x) for x in inputs] ou [function(x) for x in inputs if condition(x)]
- built-in sum — sum (range(5)) -> 10
- module operator — from operator import add, mul
- reduce — reduce (add, range(5),0) -> 10 — reduce (mul, range(1,5), 1) -> 24

- numpy — http://wiki.scipy.org/Tentative_NumPy_Tutorial
- matplotlib — http://matplotlib.org

Hidden Markov Models



- Model : λ=(A,π,E)
- decoupling state-output

E

0,1	0,2	0,7	Σ=1
0,4	0,2	0,4	

Two major kinds of Hidden Markov Model

- Discrete: Emission among a **finite** set of observations
- Continuous: Emission in 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

$$P(seq/path) = \pi(path_0) * \prod_{t=1}^{T-1} A_{path_{t-1},path_t} * \prod_{t=0}^{T-1} E_{path_t,seq_t}$$

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

  - unpracticable, 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):
  alpha[i][0] = PI[i] * E[i][seq[0]]

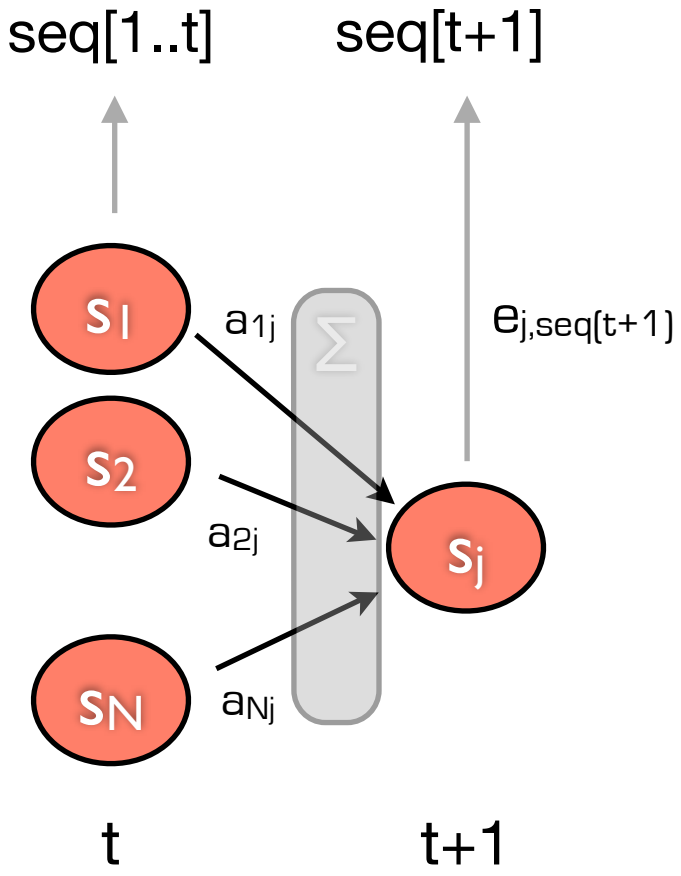
Forward - continued

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

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

- Recurrence

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



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

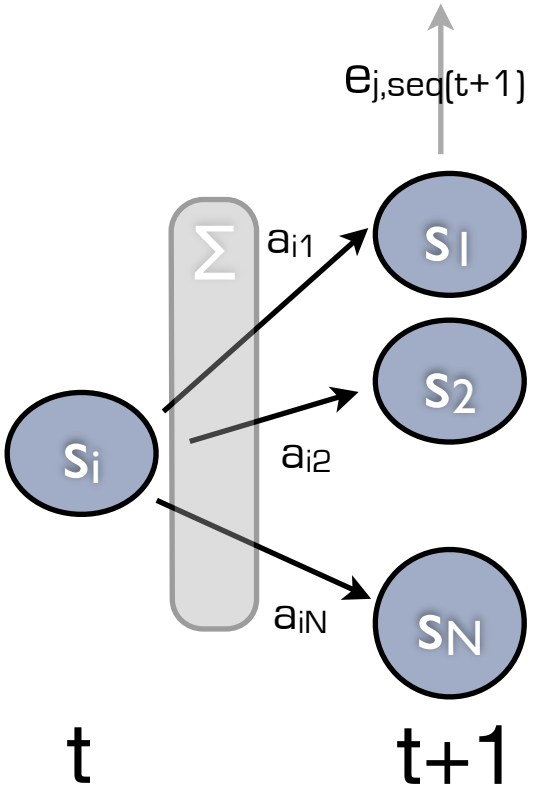
Forward - finalization

- 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**
- beta[i][T-1] = 1
- beta[i][t] = <yours to say>
- final result

$$P(seq) = \sum_{i=0}^{N-1} \pi_i * E_{i,seq_0} * beta(i, 0)$$



Implementation

- Use your locally installed python (v2 recommended ?)

- To implement a function that can be used like this

  (proba, alpha) = forward (PI, A, E, sequence)

- Same for backward

- Using numpy or not - your call

unfaircasino — a study model

2 states

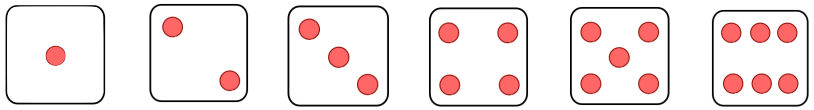
A

0,9	0,1
0,1	0,9

fair dice
biased dice

E

1/6	1/6	1/6	1/6	1/6	1/6
3/13	2/13	2/13	2/13	2/13	2/13



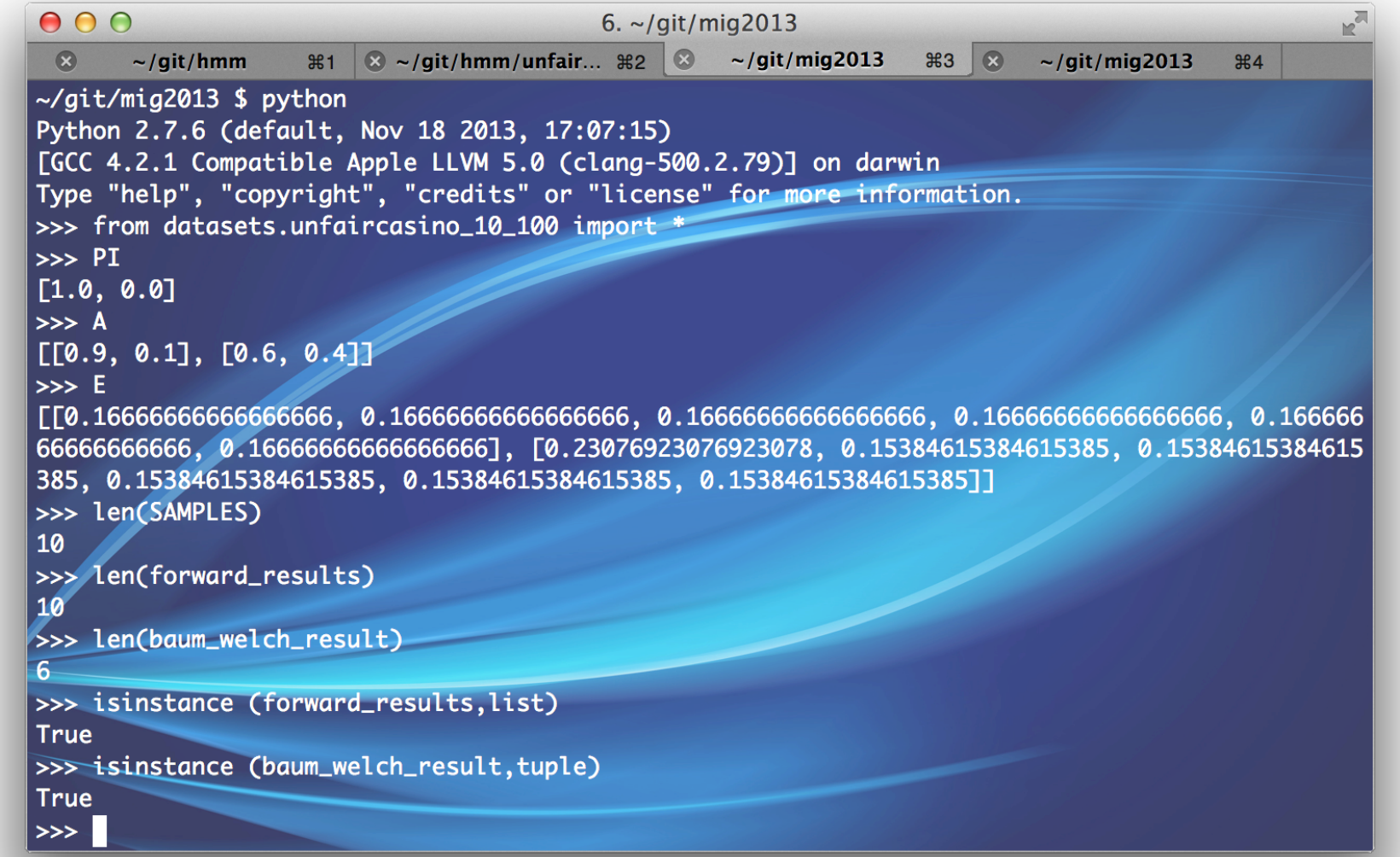
starting with the fair dice

π

1	0

Available datasets

- For comparing your results: see datasets/



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)

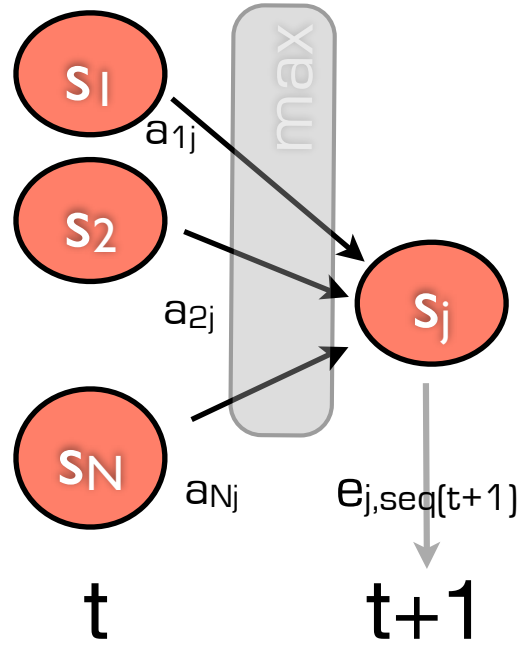
- **psi[i][t]**
  state at t-1 for the path that corresponds to *delta*
  psi[i][0] undefined / does not matter

Viterbi — details

- Initialization

  delta[i][0] = PI[i] * E[i][sequence[0]]

  psi[i][0] = None



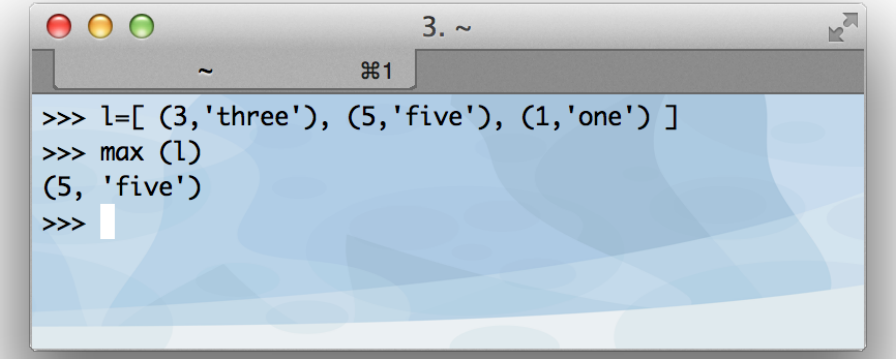
- Recurrence

  delta[j][t+1] = \

  max ([delta [i][t] * A[i][j]] for i in range(N)]) * E[j][seq[t+1]

  psi[j][t] = argmax (*same expression*)

Viterbi — a useful trick



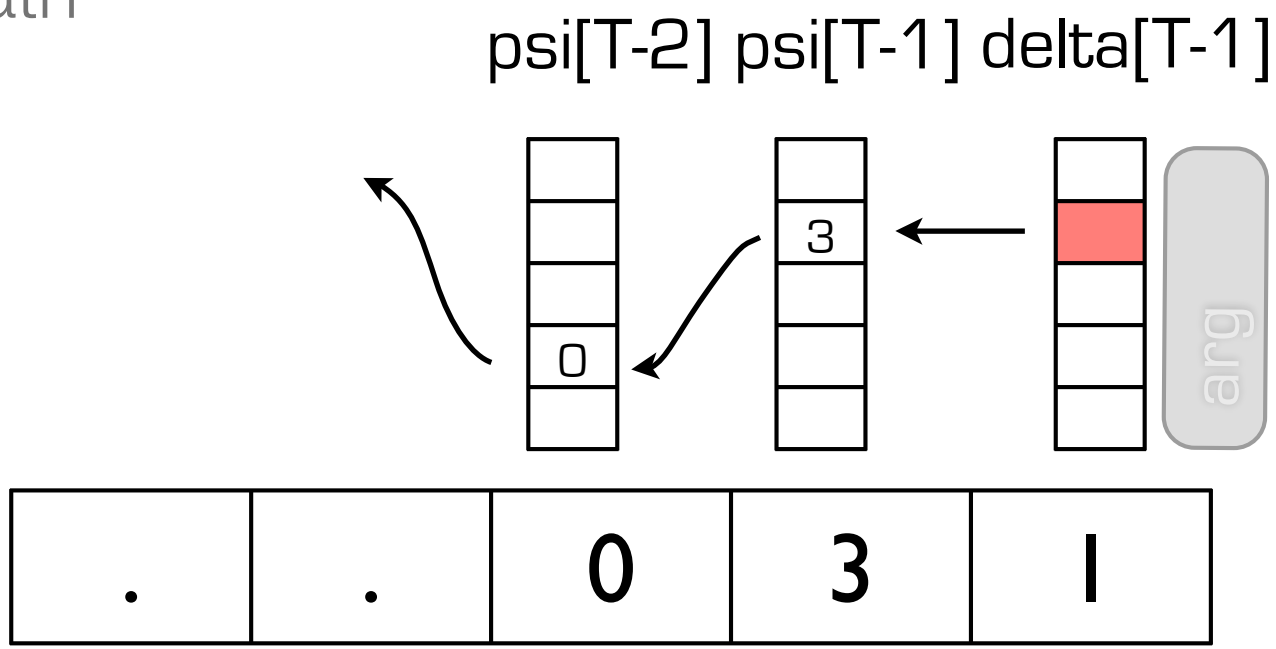
(proba, rank) = max ([(delta[i][t-1] * A[i][j] , i) for i in range(N)])

delta[j][t] = proba * E[j][seq[t]]

psi[j][t] = rank

Viterbi - finalization

- retrieve full path



- 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

  (proba, path, delta, psi) = viterbi (PI, A, E, sequence)

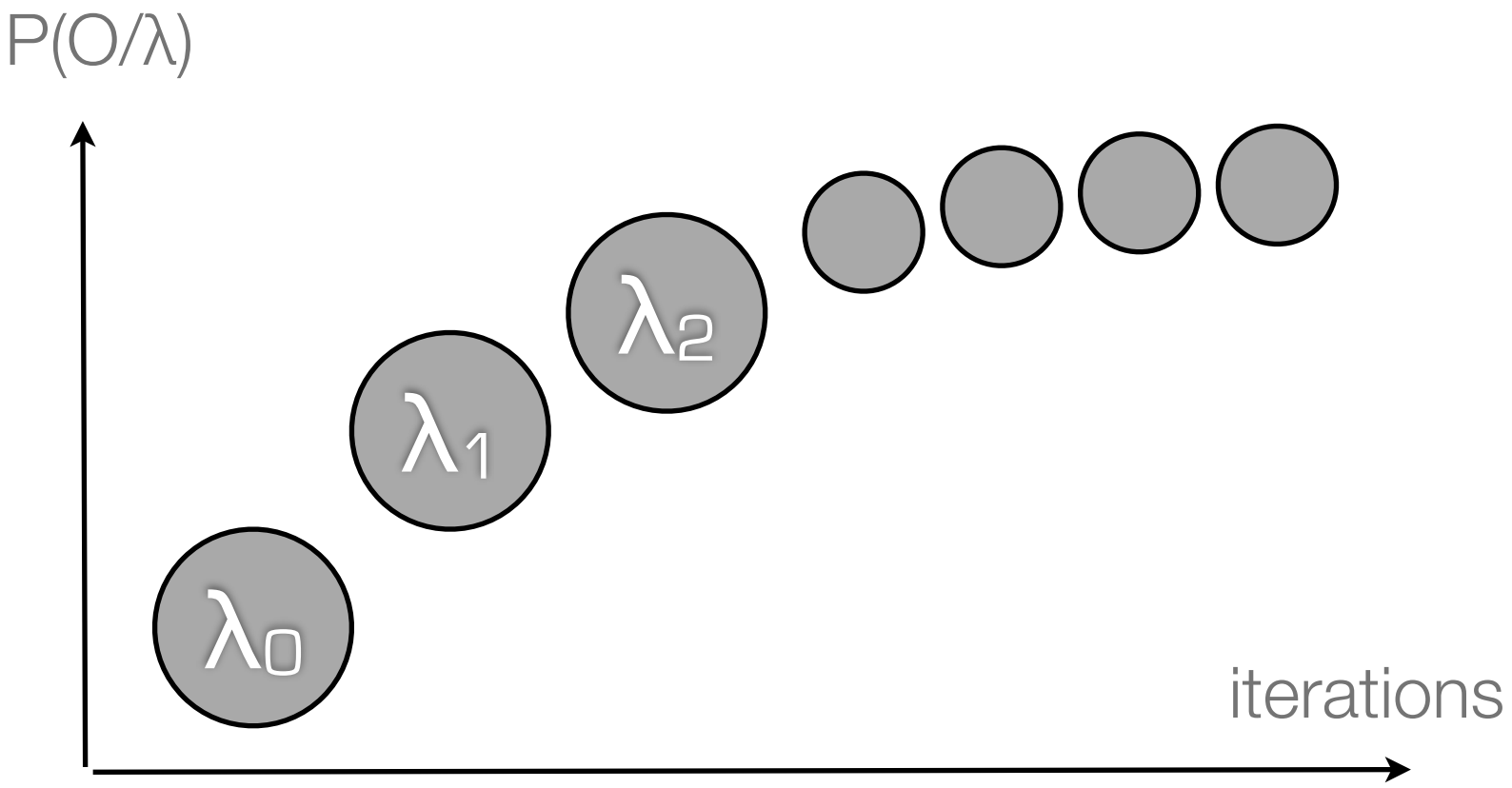
- (although delta and psi are not really useful outside)

Baum-Welch — learning

- Problem: given a set of output sequences
  Find out the “best” HMM that would give this out
- Improve model $\lambda = (A, E, PI)$ by successive iterations
- maximize likelihood over λ
  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 λ
- **xi[i][j][t]** $\xi_t[i,j]$

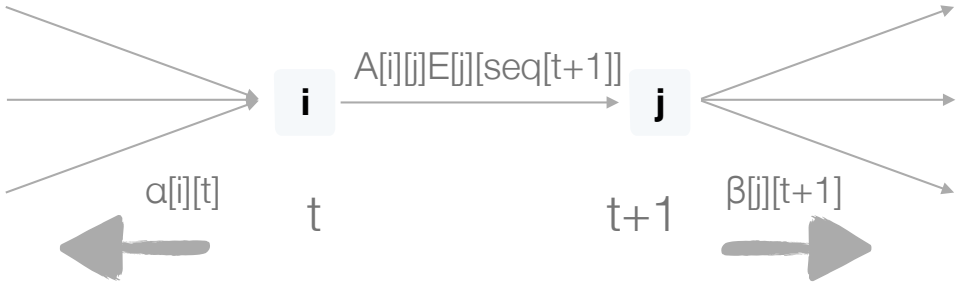
  probability to be in state **i** at time **t**, and in state **j** at **t+1**
- **gamma[i][t]** $\gamma_t[i]$

  probability to be in state **i** at time **t**

computing xi and gamma

- xi (ξ) 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 (γ) 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

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

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

- 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

$$\overline{\pi_i} = \gamma_0(i)$$

$$\overline{A_{ij}} = \frac{\textit{expected number of transitions from } \mathbf{i} \textit{ to } \mathbf{j}}{\textit{expected number of transitions from } \mathbf{i}}$$

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

$$\overline{E_{jo}} = \frac{\textit{expected number of times in } \mathbf{j} \textit{ and observing } \mathbf{o}}{\textit{expected number of times in } \mathbf{j}}$$

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

reminder from previous slides

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

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

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

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

Convergence criteria

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

Deal with multiple sample sequences

$$\overline{\pi_i} = \gamma_0(i)$$

Average on sequences

$$\overline{A_{ij}} = \frac{\textit{expected number of transitions from } \mathbf{i} \textit{ to } \mathbf{j}}{\textit{expected number of transitions from } \mathbf{i}}$$

Sum over sequences

Sum over sequences

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

$$\overline{E_{jo}} = \frac{\textit{expected number of times in } \mathbf{j} \textit{ and observing } \mathbf{o}}{\textit{expected number of times in } \mathbf{j}}$$

Sum over sequences

Sum over sequences

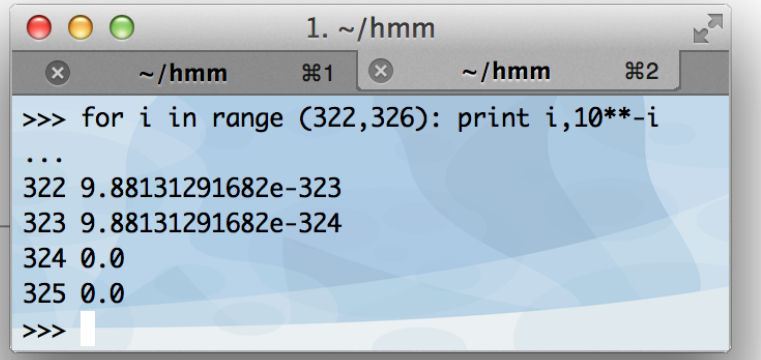
$$= \frac{\sum_{t=0}^{T-1} \gamma_t(j) \quad \textit{so that } \mathbf{o}=seq(t)}{\sum_{t=0}^{T-1} \gamma_t(j)}$$

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



- 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 $\mu = \{\mu_1, \mu_2, .. \mu_d\}$
  - a covariance matrix σ_{ij} in dimension d
- Then each state is attached a linear stochastic combination of these gaussians (a **mixture**)

Continuous HMM — example

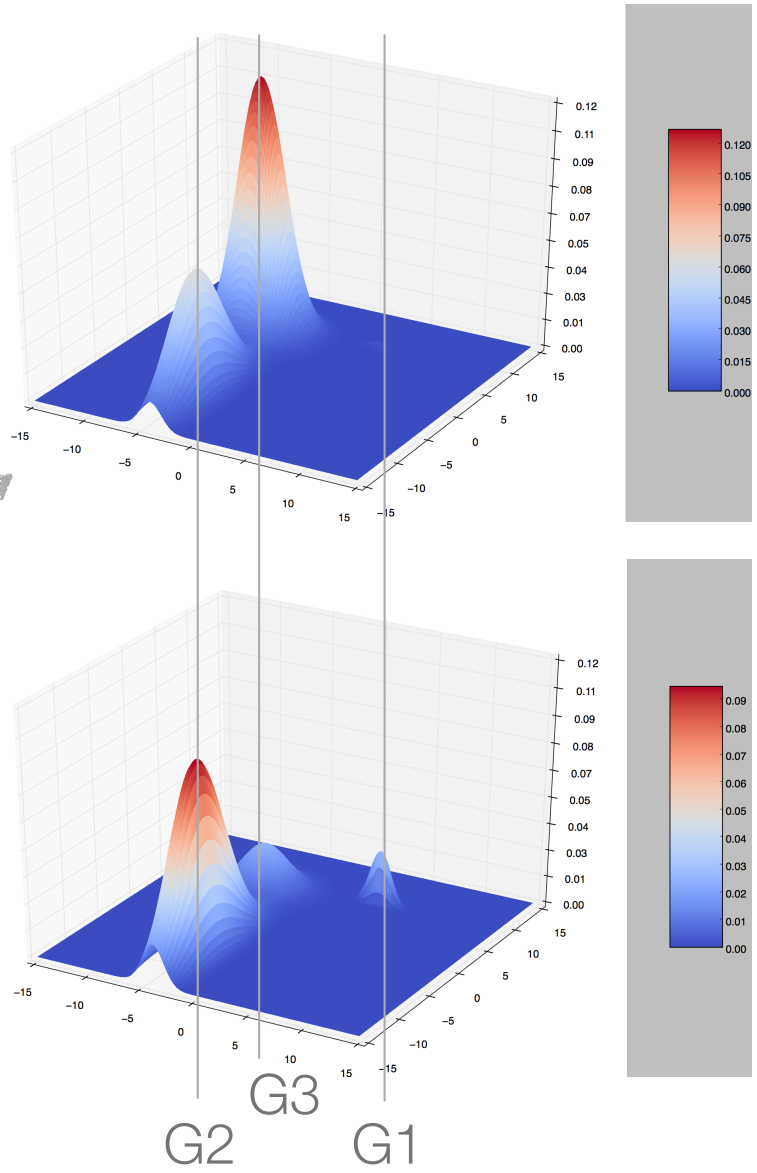
- 3 Outputs:

  G1 : μ=(4., 8.), σ=(1., 0.25)
  G2 : μ=(-4.,-8.), σ=(1., 4.)
  G3 : μ=(-6., 6.), σ=(2., 2)

  say G1~tired,
  G2~feverish, G3 ~ feels good

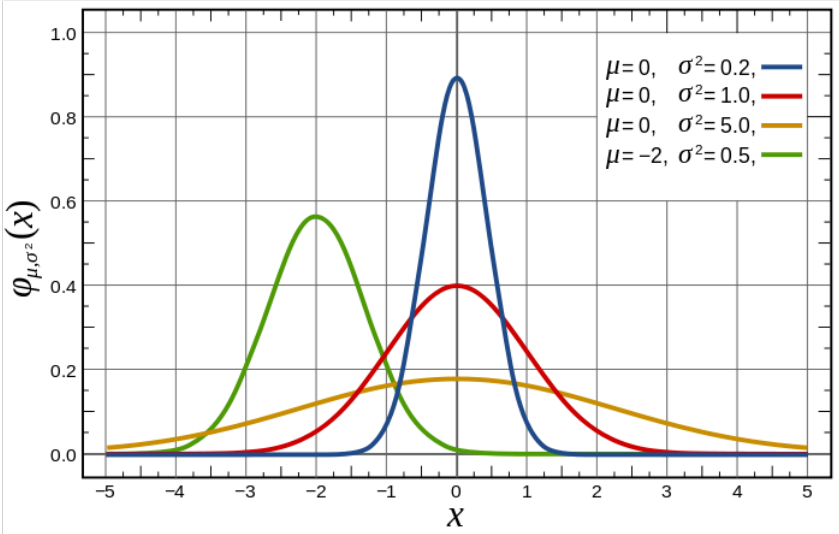
- 2 states:

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



General form of d-dimension gaussian

- a.k.a. multivariate normal distribution (wiki)
- μ average; Σ covariance
  Σ is positive semidefinite and symmetric

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

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

- check out “covariance matrix” in wiki

Degenerate case

- Diagonal covariance matrix (unrelated data)

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

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

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

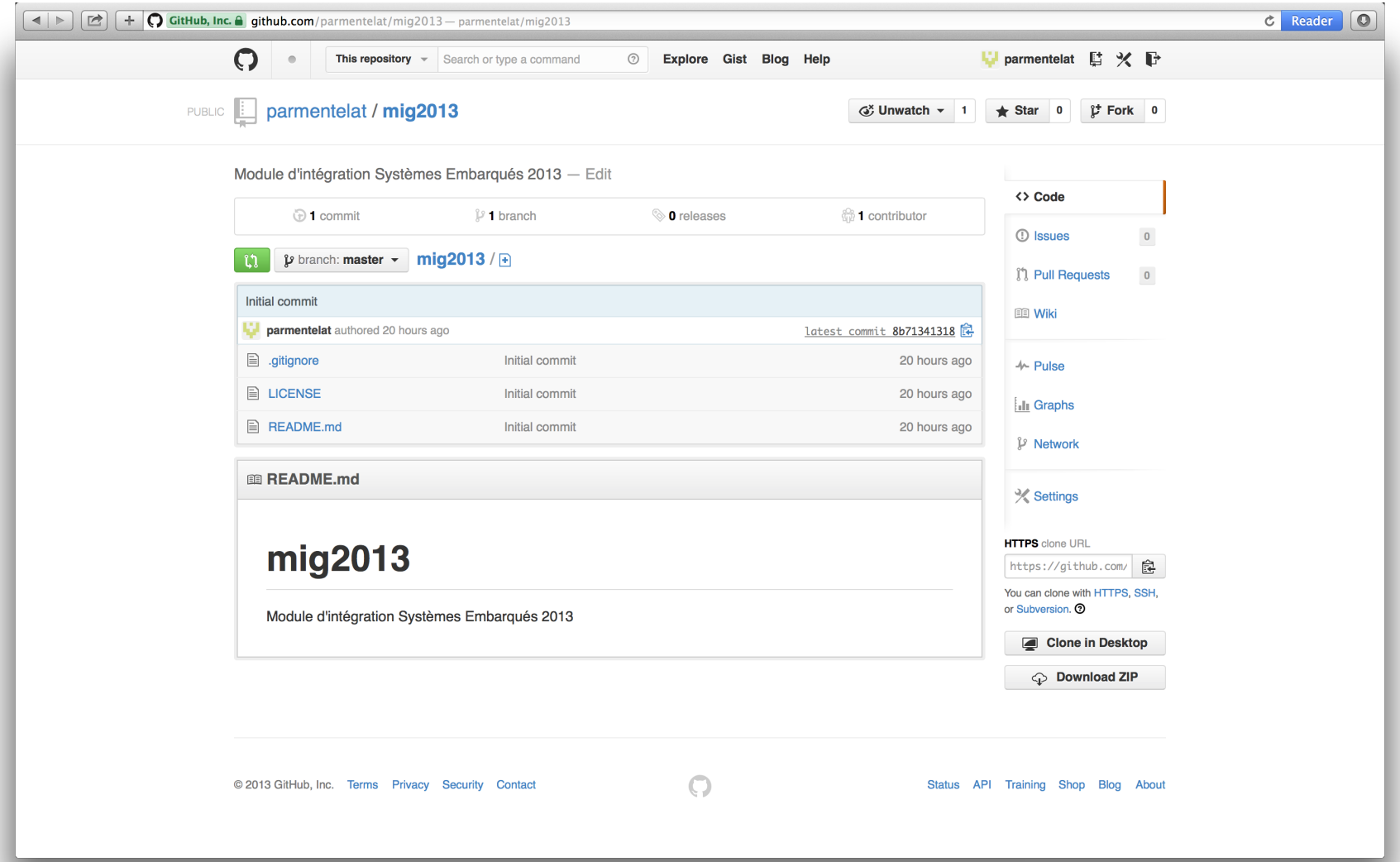
- mercurial — http://mercurial.selenic.com is nice too

git basics

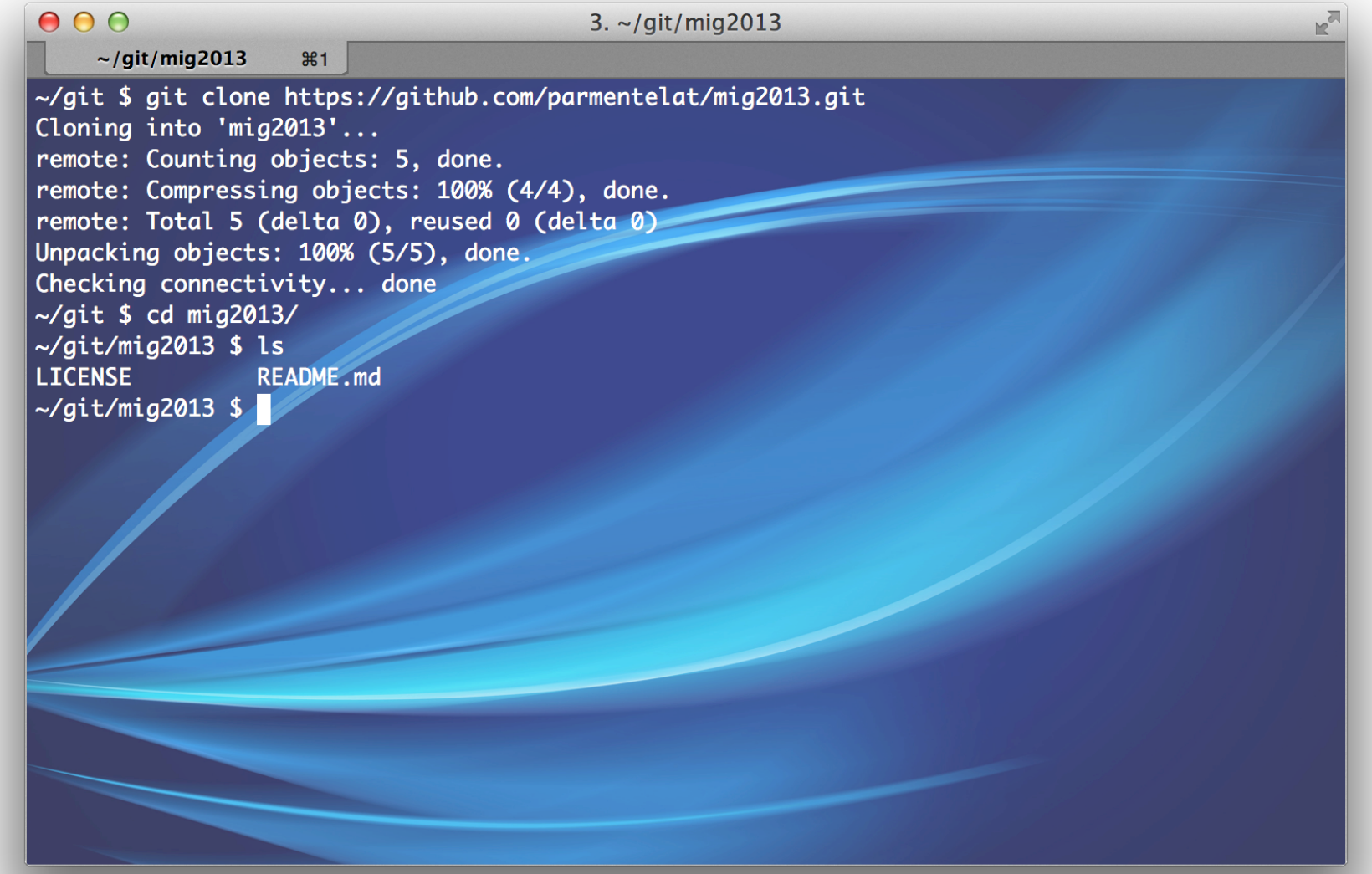
- git init — initialize git from a working directory
- git clone — create a local working dir from remote
- git add — put changes aside for next commit
- git commit — create a commit
- git pull — get changes from remote
- git push — push your commits on remote
- git stash — hide local changes under carpet

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

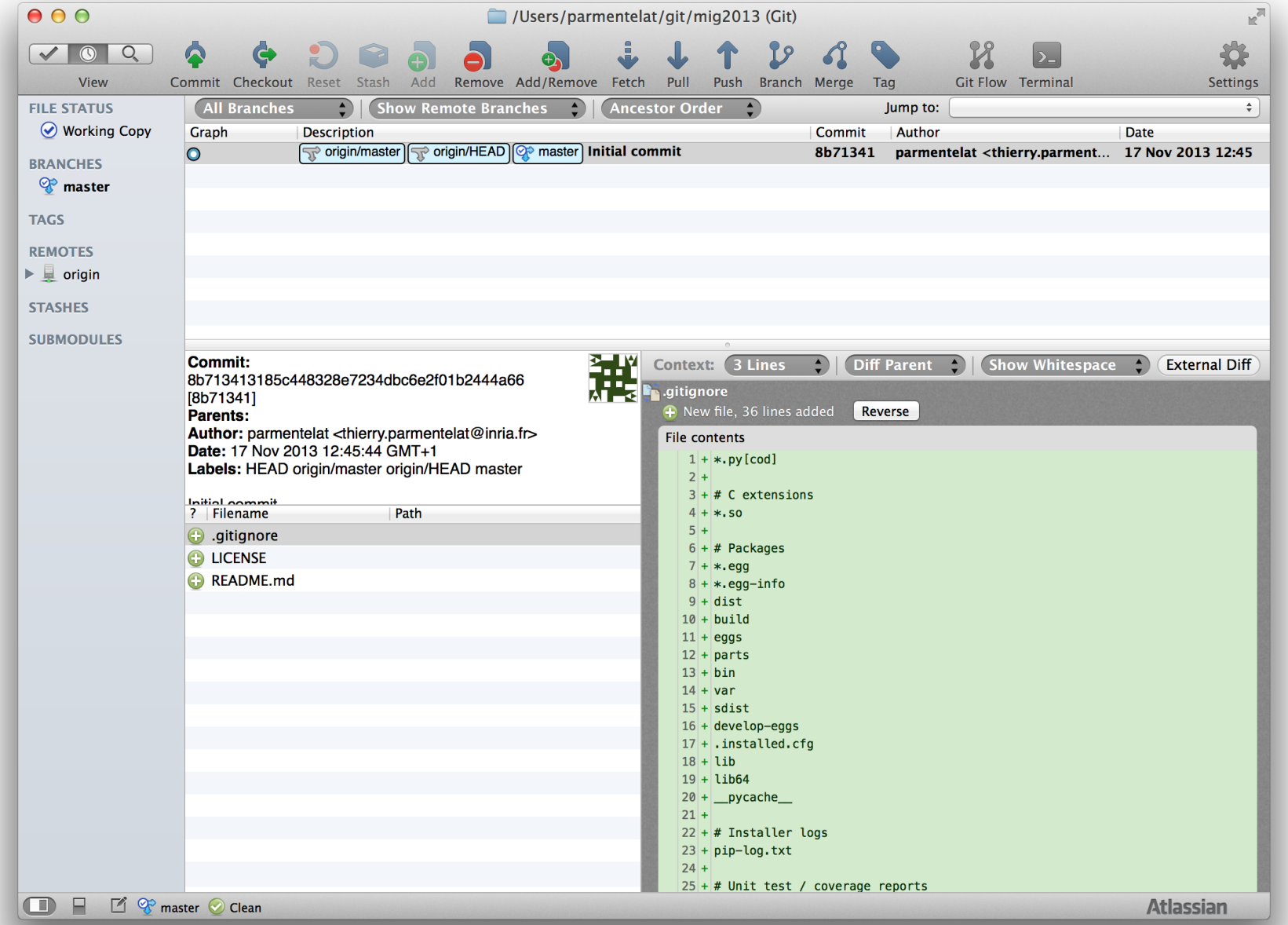
example (1) - create repo in github



example (2) - clone repo locally



example (3) - visualize local repo with sourcetree

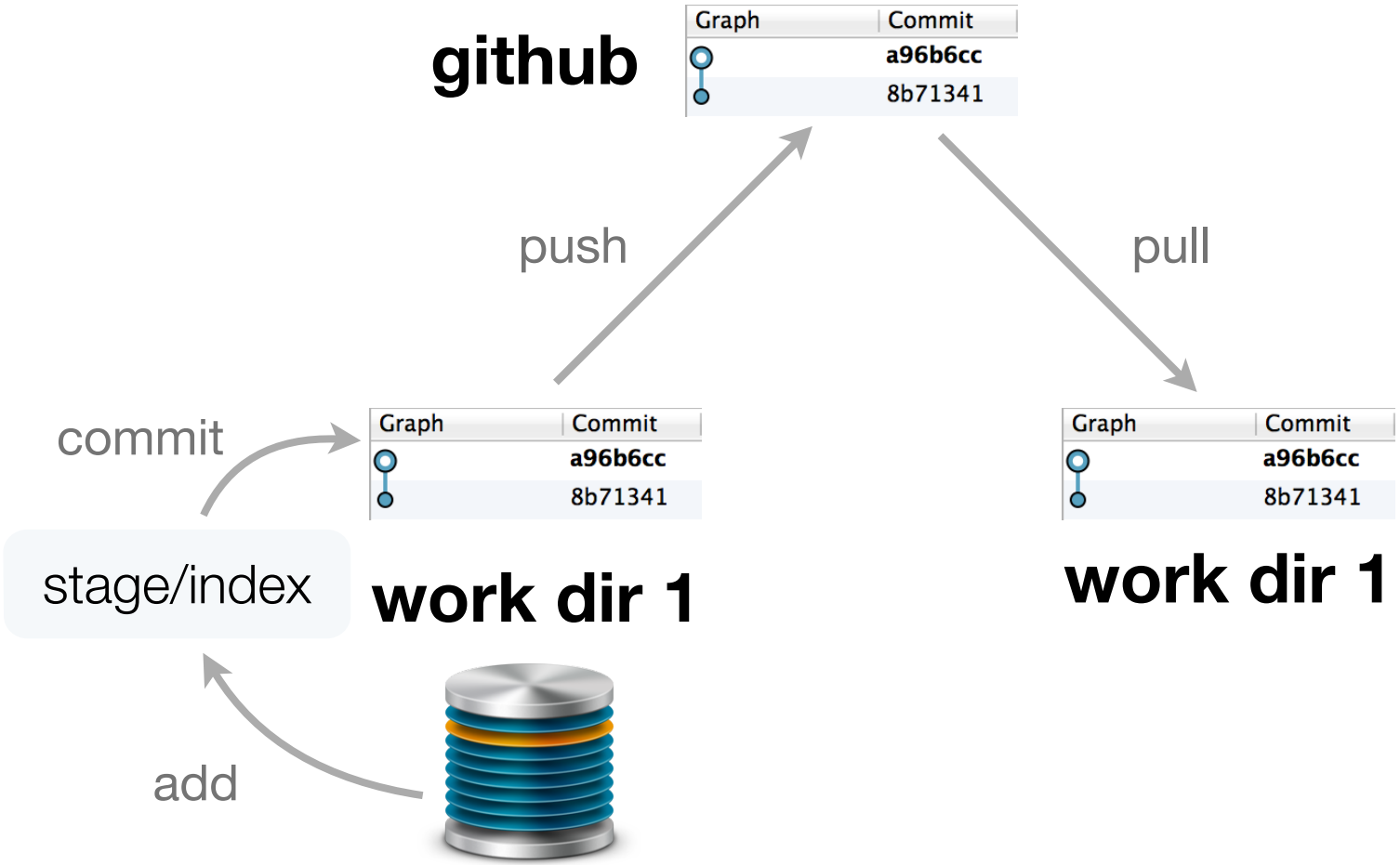


example (4) - two working directories

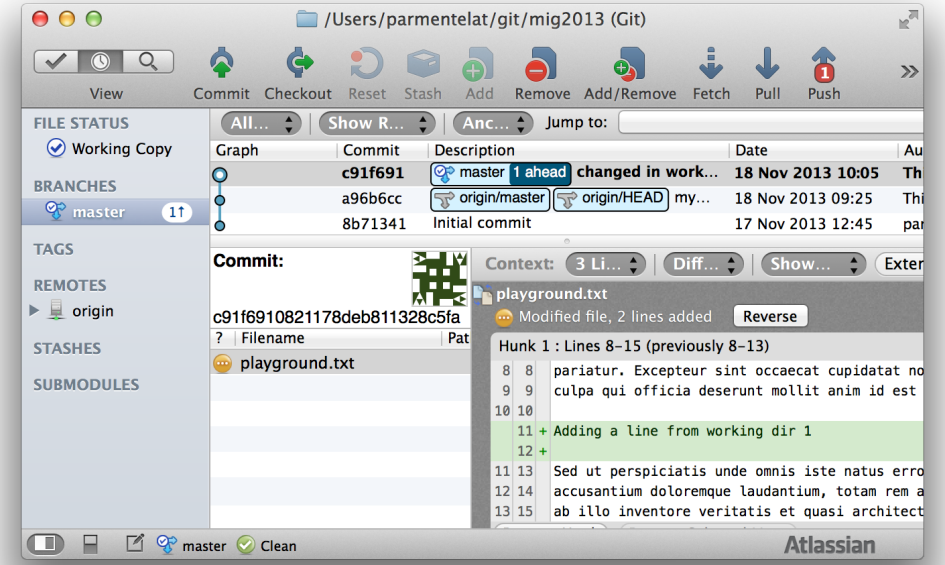
```
$ git ls-files playground.txt
$ git add playground.txt
$ git commit -m “my message”
$ git ls-files playground.txt
playground.txt
$ git push
```

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

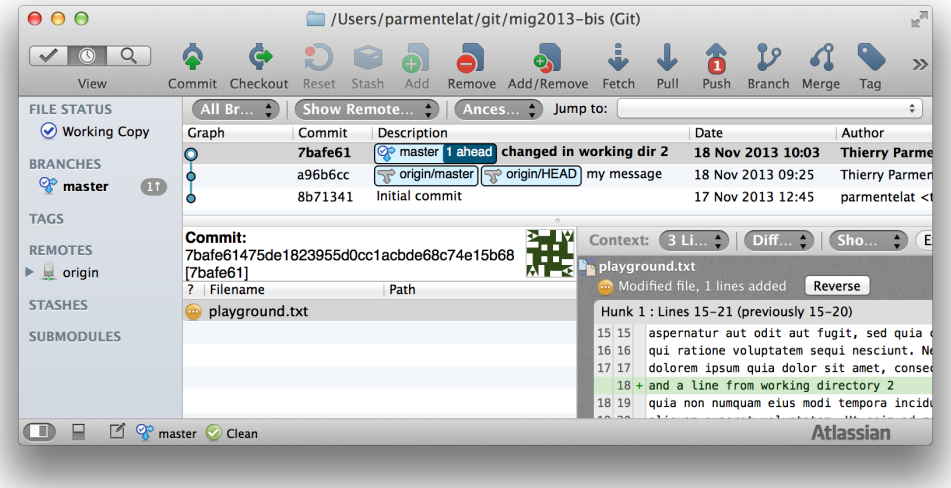
example (5) - the repositories at work



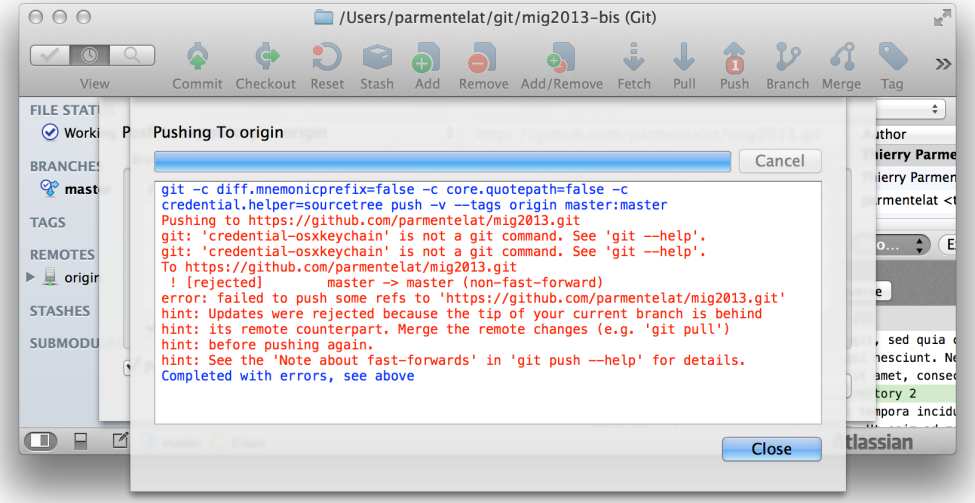
example (6) - merging two concurrent changes



user 1 can push fine

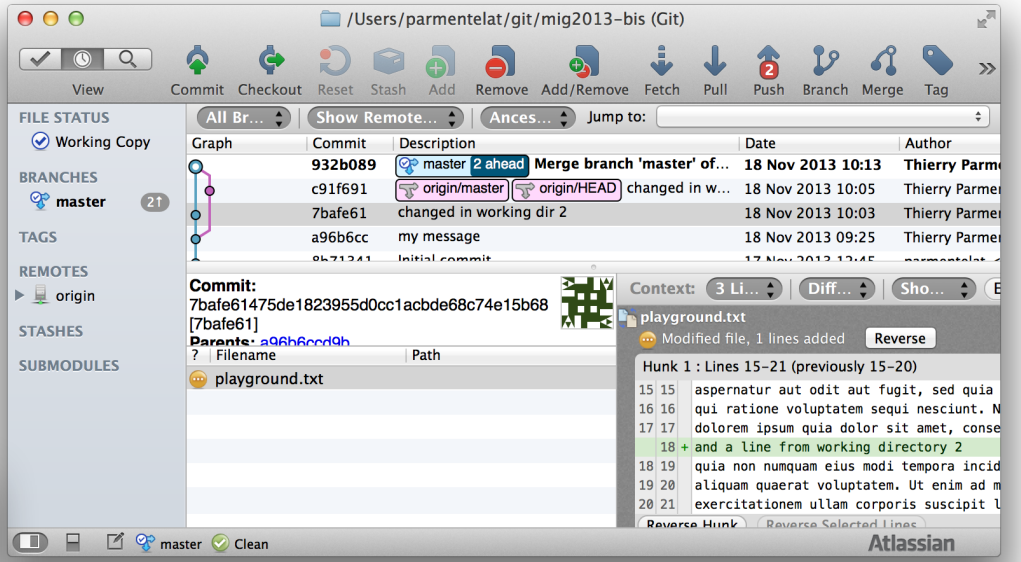


user 2 gets conflicts when pushing

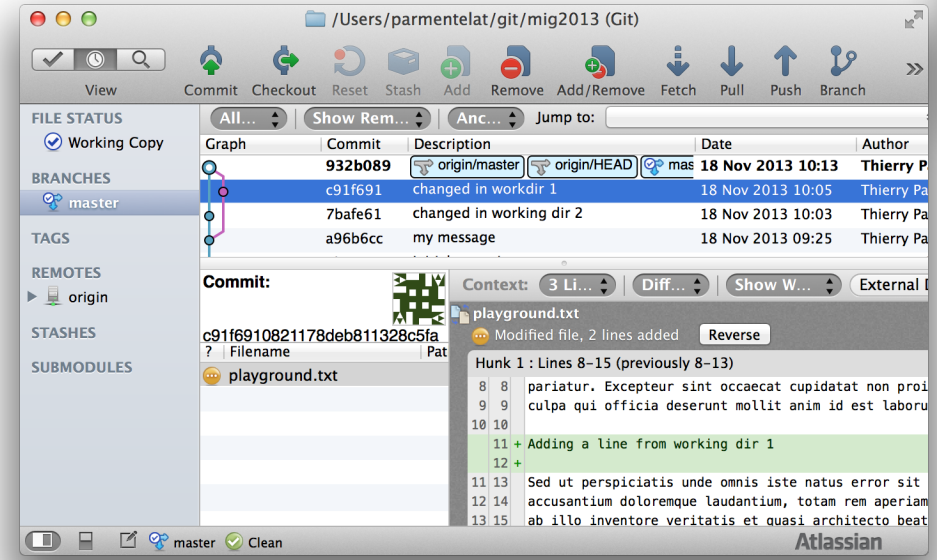


example (7) - merging - continued

user 2 needs to pull first



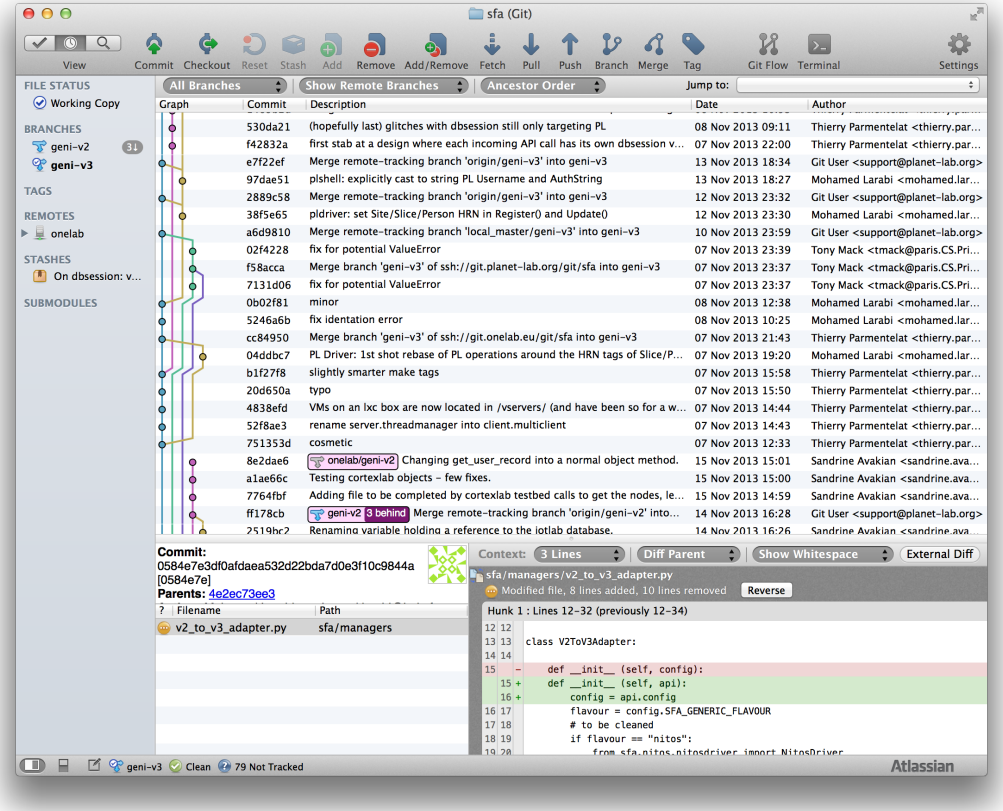
user 1 sees once he pulls again



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”



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
  - $E_{Public} \circ E_{private} = E_{private} \circ E_{public} = Identity$
  - challenge remote to “be” X (to have private key X)
  - send E_{public}(message) over the wire
  - $E_{private}$ is required to decode message