Comprendre et programmer les protocoles POP et IMAP?

L'application la plus utilisée sur l'Internet est le mèl. Pour aller lire vos messages dans votre boite aux lettres, les deux protocoles qui dominent sont POP3 et IMAP4.

Le protocole SMTP qui permet d'effectuer le transfert des emails de son origine vers ses destinataires. Il est bien rare que la boite aux lettres d'un utilisateur se trouve sur la machine sur laquelle il travaille, elle est généralement située sur un serveur dédié à cet usage. Pour y accéder, nous allons donc avoir recours à un protocole texte pour nous connecter au serveur et récupérer nos emails. Il existe en fait deux protocoles largement répandus. Le protocole POP utilisé dans sa version 3 est très simple et fournit les fonctionnalités triviales. Le protocole IMAP est plus riche et permet de travailler directement sur le serveur, la version actuelle est la 4.

Cet article se propose de vous initier au protocole POP3, aux bases du protocole IMAP, et à la programmation de clients simples en langage C et java.

Bref rappel sur SMTP

Comme tout protocole texte digne de ce nom, SMTP peut être utilisé avec la commande telnet. Pour envoyer un email nous aurons donc les commandes suivantes:

telnet smtp.wanadoo.fr 25
220 mel-rta10.wanadoo.fr ESMTP Service (6.5.007) ready
HELO salemioche.com
250 mel-rta8.wanadoo.fr
MAIL FROM: <linuxmag@salemioche.com>
250 MAIL FROM:<linuxmag@salemioche.com> OK
RCPT TO: <nicolas@salemioche.com>
250 RCPT TO:<nicolas@salemioche.com> OK
DATA
354 Start mail input; end with <CRLF>.<CRLF>
bonjour,
ceci est un message de linuxmag pour nicolas.
.
250 <3D8011E600743103> Mail accepted
QUIT
221 mel-rta7.wanadoo.fr QUIT

Nous avons ainsi un moyen simple et rapide d'envoyer un email pour tester ensuite la connexion avec les protocoles POP3 ou IMAP4.

POP3: un protocole client serveur

A l'instar des deux prédécesseurs de cette série d'article, le protocole POP3 (Post Office Protocol, version 3) est un protocole texte client-serveur. Nous nous positionnons coté client. La plupart des outils de lecture de mails supportent ce protocole, il est simple, largement répandu et propose les fonctionnalités nécessaires pour une utilisation confortable de ce support. Il est défini par la RFC 1939. Nous allons commencer par une petite session telnet qui va nous permettre de voir quasiment toutes les fonctionnalités de ce protocole.

telnet pop.wanadoo.fr 110

Le port 110 est celui généralement utilisé par le serveur pour recevoir une connexion. Vous retrouverez cette valeur si vous jetez un œil dans le fichier /etc/services (le protocole POP2, complètement obsolète aujourd'hui, utilisait le port 109). Une fois la connexion établie, le serveur s'identifie.

+OK POP3 server ready (NPlex 2.1.131) <2ea91e.1037815512000@mel-pop6.wanadoo.fr>

La réponse d'un serveur POP3 à une commande est toujours de la forme suivante: un indicateur de statut et un mot clef suivi d'un commentaire éventuel. Le statut est toujours de la forme +OK en cas de succès ou –ERR en cas d'échec. L'analyse lexicale de la réponse va donc être particulièrement aisée si l'on souhaite écrire un programme utilisant ce protocole. A présent il est nécessaire de s'authentifier en deux étapes:

USER nicolas
+OK User name accepted, password please
PASS mot_passe
+OK Mailbox open, 0 messages

C'est très simple et en plus si vous faites un telnet vers le serveur comme ici, le mot de passe apparaît en clair, il est donc impossible de se tromper. Une fois connecté et authentifié, nous avons accès à notre boite aux lettres.

LIST
+OK Mailbox scan listing follows
1 777
2 440
3 448
.

Commençons par regarder la liste des emails stockés. Pour cela on utilise la commande LIST. L'opération se passe correctement, on obtient donc le statut +OK. Sur chaque ligne on trouve le numéro du message, suivi de sa taille. La liste se termine par le '.' seul sur une ligne (ça ne vous rappelle rien ?). J'ai donc ici trois messages. Pour lire le contenu du premier message je peux faire :

RETR 1

+OK 777 octets
Return-Path: <nicolas@salemioche.com>
Delivered-To: nicolas@salemioche.servebeer.com
Received: from njhome (unknown [10.16.0.200])
by njhome.salemioche.servebeer.com (Postfix) with ESMTP id 67DA43F27F
for <nicolas@salemioche.servebeer.com>; Wed, 20 Nov 2002 13:21:50 -0500 (EST)
From: "Nicolas" <nicolas@salemioche.com>
To: <nicolas@salemioche.servebeer.com>
Subject: coucou
Date: Wed, 20 Nov 2002 19:27:45 +0100
Message-ID: <000201c290c2$856ddca0$c800100a@njhome.com>
MIME-Version: 1.0
Content-Type: text/plain;
charset="us-ascii"
Content-Transfer-Encoding: 7bit
Status: O

hello
.

Par la commande RETR, on obtient le contenu du message dont le numéro est passé en paramètre. Ce message est celui envoyé et relayé par des serveurs SMTP, il se compose donc de deux parties. L'entête est une succession de champs et de valeurs. Une ligne blanche sépare le corps du message qui se termine par un point (encore!). Pour la signification des principaux champs je vous renvoie à l'article précédent sur SMTP (les lecteurs non réfractaires à la langue de Shakespeare sauront en deviner la plupart).

Après avoir lu ce message passionnant, je peux sans hésiter le supprimer :

DELE 1
+OK Message deleted

Je vérifie (oui, j'ai pas confiance, je veux être sur de ne pas laisser de traces):

LIST
+OK Mailbox scan listing follows
2 440
3 448
.

Je remarque que les messages n'ont pas été renumérotés, on commence à présent à 2. Le message que vous avez supprimé existe toujours !! C'est seulement son entrée dans la liste qui a été supprimée, la suppression définitive n'aura lieu que au moment de la déconnection. Démonstration:

LIST
+OK Mailbox scan listing follows

1 440
2 448
.
DELE 1
+OK Message deleted
LIST
+OK Mailbox scan listing follows
2 448
.
RSET
+OK Reset state
LIST
+OK Mailbox scan listing follows
1 440
2 448
.

Tant que je suis connecté, mes messages ne sont donc pas supprimés. La commande RSET permet de les restaurer. On dit qu'ils sont marqués.

Lorsqu'un message est de taille très importante, il peut être intéressant de regarder d'abord l'entête, d'en extraire le nom de l'auteur et le sujet du message. Ainsi on peut supprimer des messages inintéressant en s'épargnant de longues minutes de téléchargement:

TOP 1 0
+OK Top of message follows
Return-Path: <root@salemioche.servebeer.com>
Delivered-To: nicolas@salemioche.servebeer.com
Received: by njhome.salemioche.servebeer.com (Postfix, from userid 0)
id 5BBC63F27F; Wed, 20 Nov 2002 13:22:34 -0500 (EST)
To: nicolas@salemioche.servebeer.com
Message-Id: <20021120182234@njhome.salemioche.servebeer.com>
Date: Wed, 20 Nov 2002 13:22:34 -0500 (EST)
From: root@salemioche.servebeer.com (root)
Status: O

.

Avec la commande TOP, je récupère l'entête du message (ici le 1) et le nombre de ligne souhaitée (ici zéro). De cette manière, je dispose de l'entête de mon message directement et rapidement. Finalement il ne me reste plus qu'à quitter la session :

QUIT
+OK Sayonara

C'est à ce moment, en quittant la session que les messages marqués à l'aide de la commande DELE seront effectivement supprimés.

Les états

Dans une session de connexion, on distingue trois états. Le premier est la phase d'authentification, le deuxième est la phase de transaction et enfin la mise à jour. A chacune de ces étapes un certain nombre de commandes sont disponibles.

Authentification

La phase d'authentification est très simple. On doit utiliser les commandes USER et PASS pour respectivement s'identifier et s'authentifier. On peut également quitter immédiatement la session à l'aide de la commande QUIT. Les plus paranos d'entre nous, ou les lecteurs de l'excellent numéro HS n°12 de linux magazine, auront noté que le mot de passe est transmis en clair. Cela est certes mieux qu'avec POP2, pas de mot de passe du tout, mais on est encore loin de ce que l'on peut espérer aujourd'hui. Il existe donc une commande optionnelle, et donc pas supportée par tous les serveurs, APOP. Cette commande permet l'authentification avec un encryptage MD5:

APOP login md5_digest

Le digest est calculé à partir de la concaténation de la clef fournit par le serveur lorsqu'il répond pour la première fois (*<2ea91e.1037815512000@mel-pop6.wanadoo.fr)* et du mot de passe.

> echo –n "<2ea91e.1037815512000@mel-pop6.wanadoo.fr>mot_passe" | md5sum
e7a0a8acb857d0b1e99c4a9974c00d0d -

je n'aurais donc plus qu'à m'authentifier avec la commande suivante:

APOP nicolas e7a0a8acb857d0b1e99c4a9974c00d0d

Transaction

Nous avons vu déjà plusieurs commandes permettant de travailler pendant la phase de transaction. Voici la liste complète:

Commandes POP3 de Transaction	
Commande	Description
STAT	Information sur les messages contenus sur le serveur
RETR n	Lire le message n
DELE n	Marquer le message n à supprimer
LIST [n]	Afficher la liste des messages ou le message n
TOP n X	Affiche X lignes du message n (commande optionnelle)
QUIT	Passer en mode mise à jour et terminer la session
NOOP	Permet de garder la connexion ouverte

Mise à jour

La saisie de la commande QUIT permet d'entrer dans la phase de mise à jour du serveur. C'est à ce moment que les messages marqués 'à supprimer' seront réellement détruits. De ce fait, si votre session 'tombe' avant que vous ayez atteint cette phase, vos messages marqués ne seront pas détruits et vous les retrouverez à votre connexion suivante.

De théorie assez à C

Voyons a présent comment accéder à notre boite aux lettres avec un petit programme en langage C. La gestion des erreurs a volontairement été supprimée afin de simplifier au maximum le code :

```
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PACKET_SIZE  1024

int writen(int fd, char *ptr, int n);
int readn(int fd, char *ptr, int n);
int test_OK(char *buf, int do_exit);

char *server_name = "NOM de votre SERVEUR POP3";
unsigned short server_port=110;

char *user= "USER de la boite";
char *pass= "MOT DE PASSE";

int to_server_socket = -1;

int main (int argc, char *argv[])
{
struct sockaddr_in serverSockAddr;    /* addresse de la socket */
struct hostent *serverHostEnt;        /* description du host serveur */
unsigned long hostAddr;               /* addr du serveur */
unsigned char bufw[PACKET_SIZE+1];
unsigned char bufr[PACKET_SIZE+1];
unsigned char list[PACKET_SIZE+1];
int n, msg, index, retry = 4;
```

```
bzero(&serverSockAddr,sizeof(serverSockAddr));
hostAddr = inet_addr(server_name);


if ( (long)hostAddr != (long)-1)
       bcopy(&hostAddr,&serverSockAddr.sin_addr,sizeof(hostAddr));
else                        /* si on a donne un nom  */
{
   serverHostEnt = gethostbyname(server_name);
   bcopy(serverHostEnt->h_addr,&serverSockAddr.sin_addr,
        serverHostEnt->h_length);
}

serverSockAddr.sin_port = htons(server_port);
serverSockAddr.sin_family = AF_INET;
/* creation de la socket */

to_server_socket = socket(AF_INET,SOCK_STREAM,0);
/* requete de connexion */
connect(to_server_socket,(struct sockaddr *)&serverSockAddr,
         sizeof(serverSockAddr));

bzero(bufw, PACKET_SIZE+1);
bzero(bufr, PACKET_SIZE+1);

do {
  n=readn(to_server_socket,bufr,PACKET_SIZE);
  retry--;
} while(test_OK(bufr,retry==0));

/* authentification */
sprintf(bufw,"USER %s\r\n",user);
writen(to_server_socket,bufw,strlen(bufw));
n=readn(to_server_socket,bufr,PACKET_SIZE);
test_OK(bufr,1);

sprintf(bufw,"PASS %s\r\n",pass);
writen(to_server_socket,bufw,strlen(bufw));
n=readn(to_server_socket,bufr,PACKET_SIZE);
test_OK(bufr,1);

/* liste des messages dans la boite aux lettres */
sprintf(bufw,"LIST\r\n");
writen(to_server_socket,bufw,strlen(bufw));
n=readn(to_server_socket,bufr,PACKET_SIZE);
test_OK(bufr,1);
```

```
  if (strlen(bufr) == 0 )
    n=readn(to_server_socket,list,PACKET_SIZE);
  else
    bcopy(bufr, list, strlen(bufr) + 1) ;

  /* recupération de tous les messages */
  index = 0;
  while ( list[index] != '.' ) {

    sscanf(&list[index], "%d", &msg);
    while(list[index++] != '\n');

    /* recuperation du message msg */
    sprintf(bufw,"RETR %d\r\n",msg);
    writen(to_server_socket,bufw,strlen(bufw));
    do {
      n=readn(to_server_socket,bufr, PACKET_SIZE);
      printf("%s",bufr);
      if ( ! strncmp("\r\n.\r\n",&bufr[n-5],5) ) break;
      bzero(bufr, PACKET_SIZE+1);
    } while ( 1 );

  }

  /* fermeture de la connection */
  shutdown(to_server_socket,2);
  close(to_server_socket);
  return 0;
}


int writen(int fd, char *ptr, int n)
{
int nl, nw;
  nl = n;
  while ( nl > 0 ) {
        nw = write(fd, ptr, nl);
        if ( nw <= 0 )
              return nw;     /*error*/
        nl -= nw;
        ptr += nw;
  }
  return (n-nl);
}

int readn(int fd, char *ptr, int n){
int nl, nr;
```

```
 nl = n;
 while ( nl > 0 ) {
  nr = read(fd,ptr,nl);
  if (nr < 0 ) return nr;     /*erreur*/
  else if ( nr == 0 ) break;
  nl -= nr;
  ptr += nr;
  if ( *(ptr-2) == '\r' &&  *(ptr-1) == '\n' )
    break;
 }
 *ptr = 0x00;
 return (n-nl);
}

int test_OK(char *buf, int do_exit) {
char *ptr, tmp[PACKET_SIZE+1];

  bzero(tmp, PACKET_SIZE+1);
  if ((ptr=strstr(buf, "+OK")) == NULL) {
      if ( strstr(buf, "-ERR") ) {
        printf("ERROR: -->%s<--\n", buf);
        exit(1);
      }
    if (do_exit ) exit(1);
    else return 1;
  } else {
    while(*ptr != '\n') ptr++ ;
    bcopy(ptr+1,tmp,strlen(ptr));
    bzero(buf,PACKET_SIZE+1);
    bcopy(tmp, buf, strlen(tmp));
  }
  return 0;
}
```

On définit trois variables qui sont "server_name", le nom de votre serveur POP3, "user", le nom du compte et "pass" son mot de passe. Modifier ces variables pour tenir compte de votre configuration. Il suffit ensuite de compiler ce programme tout simplement :

```
 gcc –o client client.c
```

et de le lancer:

client

La première partie du programme est dédiée à la connexion avec le serveur, ensuite le dialogue s'installe et l'on retrouve la succession de commandes/réponses du protocole. Une fois authentifié, on récupère la liste des messages à l'aide de la commande RETR et on affiche leur contenu sur la

console. Dans l'exemple ci-dessus, les messages ne sont pas supprimés. Pour ce faire, il faut ajouter la commande DELE après la lecture du message.

Et en java

A présent la même chose en java:

```
import java.io.*;
import java.net.*;
import java.util.*;

class Pop {
String server;
String user;
String pass;

    Pop(String zserver, String zuser, String zpass) {
      server = zserver;
      user = zuser;
      pass = zpass;
    }

    void lit() {
    PrintWriter to;
    BufferedReader from;
    String str,title, msg;
    Vector v = new Vector();

     try {
      Socket socket = new Socket(InetAddress.getByName(server),110);

      to = new PrintWriter(
                new BufferedWriter (
                   new OutputStreamWriter (
                      socket.getOutputStream())),true);
      from = new BufferedReader(
                new InputStreamReader (
                      socket.getInputStream()));

      while ( ! (from.readLine()).startsWith("+OK") );
      to.println("USER "+user+"\r");
      while ( ! (from.readLine()).startsWith("+OK") );
      to.println("PASS "+pass+"\r");
      while ( ! (from.readLine()).startsWith("+OK") );
```

```
      to.println("LIST\r");
      while ( ! (from.readLine()).startsWith("+OK") );
      do {
        str = from.readLine();
        if ( str.compareTo(".") != 0 ) v.add(str);
      } while ( str.compareTo(".") != 0 );

      for ( int i =0; i < v.size(); i++ ) {
        title = (String ) v.elementAt(i);
        to.println("RETR "+(new StringTokenizer(title)).nextToken()+"\r");
        while ( ! (from.readLine()).startsWith("+OK") );
        msg = "";
        do {
          msg += from.readLine() + "\n";
        } while ( ! msg.endsWith("\n.\n") );
        System.out.println(msg);
      }

      socket.close();

    } catch ( Exception e ) {System.err.println(e);}
  }


  public static void main (String args[]) {
    Pop p = new Pop("serveur","user","pass");
    p.lit();
  }
}
```

Vous obtiendrez donc le même résultat qu'avec le programme en C, après avoir encodé:

javac Pop.java
puis exécuté cette classe:

java Pop

N'oubliez pas d'instancier cette classe avec la configuration de votre compte pop.

Zap sur IMAP

Contrairement à POP, l'objectif de IMAP (Internet Message Access Protocol) est la manipulation des emails et des boites aux lettres 'en ligne'. Avec POP, on récupère sur le serveur les messages qui sont ensuite stockés sur la machine cliente afin d'être consultés et traités, au contraire avec

IMAP on travaille directement sur le serveur. Si les messages peuvent bien sur être téléchargés sur la machine, ce n'est pas le but.

Bien que ce protocole ait été pour la première fois défini en 1986, c'est seulement depuis 1998 qu'il a été spécifié par la RFC 1064. Aujourd'hui on utilise la révision 1 de la version 4 d'IMAP définie par la RFC 2060 sous le nom IMAP4rev1.

Le fait d'accéder à la boite aux lettres sur le serveur offre les avantages suivants:

- Possibilité d'utiliser plusieurs machines à différents instants.
- Possibilité d'utiliser des machines clientes "sans-données" (c'est aussi le principe des webmails).
- Accès à de multiples boîtes aux lettres indépendamment de la plate-forme.
- Possibilité d'accès concurrentiel à des boîtes aux lettres partagées (très pratique pour une équipe de support/hotline par exemple).

Les messages arrivés sont stockés dans un dossier nommés INBOX mais IMAP permet de définir d'autres dossiers (et les opérations standards création/suppression/modification sont autorisées). Cela permet de classer ses messages de façon logique et hiérarchique.

IMAP offre l'usage de flags (fanions/drapeaux) pour marqués les messages en fonction de leur état (par exemple: \Seen, \Answered, \Flagged, \Deleted, \Draft \Recent).

Afin de limiter la bande passante utilisée pour le transfert d'un message, il est possible de récupérer sa structure (expediteur, destinataire…) sans le corps du message, ce qui permet de filtrer directement sur le serveur sans avoir à rapatrier tout l'email sur le client. De même on peut pour les messages complexes avec plusieurs attachements ne lire qu'un de ceux-ci.

Nous venons de voir les quelques avantages qu'offre le protocole IMAP, la contre partie vous vous en doutez, c'est que ce protocole est beaucoup plus riche et complexe que son rival POP.

IMAP: les bases du protocole

IMAP est donc un protocole client serveur utilisant généralement une connexion TCP sur le port 143 du serveur. Une connexion IMAP comprend d'abord l'établissement d'une connexion réseau, puis un message de salutation initial du serveur et enfin les interactions client/serveur. Ces interactions client/serveur consistent en une suite de commandes/réponses entre le client et le serveur. Toutes les données transmises entre le client et le serveur se présentent sous forme de lignes qui se terminent par CRLF ('\r\n' en langage C ou java). Rien de neuf donc.

Une commande du client commence une opération. Chaque commande du client est préfixée par un identificateur (typiquement une chaîne alphanumérique courte, par exemple : A0001, A0002, etc.) appelé tag. Un tag différent est généré par le client pour chaque nouvelle commande. Le serveur transmet les données appropriées ainsi qu'une réponse, résultat de commande complétée.

Session telnet

Voyons quelques unes des commandes de ce protocole à travers une session telnet. La liste des possibilités est trop longue pour cet article et une fois l'eau à la bouche, je vous laisse consulter la RFC pour tous les détails. C'est parti:

telnet mon_serveur 143

le serveur répond:

** OK [CAPABILITY IMAP4REV1 LOGIN-REFERRALS AUTH=LOGIN] mon_serveur IMAP4rev1 2001.315 at Tue, 26 Nov 2002 18:46:22 +0100 (CET)*
a1 login nicolas mot_passe

on s'authentifie de façon classique. On notera la *a1* devant la commande.

a1 OK [CAPABILITY IMAP4REV1 IDLE NAMESPACE MAILBOX-REFERRALS SCAN SORT THREAD=REFERENCES THREAD=ORDEREDSUBJECT MULTIAPPEND] User nicolas authenticated

Le serveur répond en préfixant son message avec le tag de ma commande (ici *a1*) et donne ensuite un statut, OK si tout va bien, NO si la commande a échoué et enfin BAD en cas d'erreur de protocole (tel qu'une mauvaise syntaxe de la commande). Il est temps de sélectionner notre boite:

a2 select inbox
** 3 EXISTS*
** 1 RECENT*
** OK [UIDVALIDITY 1038331903] UID validity status*
** OK [UIDNEXT 4] Predicted next UID*
** FLAGS (\Answered \Flagged \Deleted \Draft \Seen)*
** OK [PERMANENTFLAGS (* \Answered \Flagged \Deleted \Draft \Seen)] Permanent flags*
** OK [UNSEEN 2] first unseen message in /var/mail/salemio*
a2 OK [READ-WRITE] SELECT completed

Les informations renvoyées sont précédées d'une étoile. On peut lire qu'il y a trois messages existants dans cette boite aux lettres, un seul est arrivés depuis la dernière connexion et deux (dernière ligne) n'ont pas été lus. Continuons.

a3 fetch 1:3 RFC822.SIZE
** 1 FETCH (RFC822.SIZE 889)*
** 2 FETCH (RFC822.SIZE 1756)*
** 3 FETCH (RFC822.SIZE 890)*
a3 OK FETCH completed

FETCH est la commande permettant de récupérer des informations sur un message. 1:3 permet de spécifier 'du message 1 au message 3', j'aurais aussi pu saisir '1:*' ce qui aurais eu le même résultat mais qui est plus générique. La commande FETCH prend donc en paramètres le message à traiter et l'information que l'on veut récupérer (ici la taille). Il est également possible de récupérer des informations dans l'entête du message:

a5 fetch 1 BODY[HEADER.FIELDS (from)]
** 1 FETCH (BODY[HEADER.FIELDS ("FROM")] {46}*
From: root@salemioche.servebeer.com (root)

)
a5 OK FETCH completed

C'est un peu plus compliqué. On choisi la section HEADER.FIELDS (c'est la liste des champs de l'entête) dans le message et on ne sélectionne que le champ 'from' (expéditeur). La liste des paramètres que l'on peut saisir pour la commande FETCH se trouve dans la RFC du protocole IMAP et le format des champs que l'on peut sélectionner est détaillé dans la RFC822 (mais on s'en sort généralement très bien avec seulement from, to et date). Récupérons quand même un message complet:

a6 fetch 1 BODY[HEADER]
** 1 FETCH (BODY[HEADER] {883}*
Return-Path: <root@salemioche.servebeer.com>
Received: from njhome.salemioche.servebeer.com
by sidonie.nfrance.com (8.11.6/8.11.6/NFrance Conseil Antispam Version) with ESMTP id gAQHOvq96585
for <nicolas@salemioche.com>; Tue, 26 Nov 2002 18:24:57 +0100 (CET)
To: nicolas@salemioche.com
Date: Tue, 26 Nov 2002 12:30:35 -0500 (EST)
From: root@salemioche.servebeer.com (root)

)
a6 OK FETCH completed
a7 FETCH 1 BODY[TEXT]
** 1 FETCH (BODY[TEXT] {6}*
toto
)
a7 OK FETCH completed
Il est temps de partir:
a10 logout
** BYE mon_serveur IMAP4rev1 server terminating connection*
a10 OK LOGOUT completed

Il est donc possible de récupérer des informations très précises sur vos messages avec ce protocole. Agir contre le spam, supprimer directement vos messages sur le serveur avant de les charger sur votre machine, limiter votre temps de connexion, partager une boite aux lettres,

fournir une interface webmail, ce n'est pas les applications qui manquent pour tirer partie des fonctionnalités d'IMAP.

IMAP ou POP

Si vous disposez d'une ligne haut débit pour votre accès Internet et d'un unique poste sur lequel vous travaillez, IMAP ne vous sera sans doute pas d'une grande utilité. Mais dans tout les autres cas, il peut vous procurez le confort d'un accès rapide à votre boite aux lettres, une sélection fine des messages à charger et à supprimer, une protection contre les virus chargés dans des emails… Pour le développement d'outils spécifiques tel la gestion d'une mailing liste, pourquoi récupérer des messages entiers avec POP3 alors qu'il peut suffire de lire le sujet de tous les messages d'un coup avec IMAP. Il est très simple d'activer le service d'accès POP3 ou IMAP sous Linux, alors choisissez celui qui vous simplifiera la vie.

Conclusion

La récupération d'emails sur un serveur distant n'a plus de secret pour vous. Que vous préfériez POP ou IMAP, le développement sera toujours simple et confortable grâce au format texte de ces protocoles. Il nous reste à étudier le format des emails avec attachements et l'encodage MIME pour finir cette trilogie sur le courriel.

Nicolas JEAN

http://www.nicolasjean.com