1, Linus的 git tree，AKA 'mainline kernel':  
  
        $ git -clone git :// git .kernel.org/pub/scm/linux/kernel/ git /torvalds/linux-2.6. git   
  
   其他的 git trees可以在[url]http:// git .kernel.org/[/url]找到，点击进去就能看到 GIT URL。  
  
  
2, 安装 git docs  
  
   要有asciidoc、 wish 命令，安装tcl/tk和asciidoc. wish 命令在tk-x.xx.xx包中。  
  
   编译 git 时，  
            
        $ sudo make install-doc  
  
  
3, 使用 git   
  
   $ git -config user.name "Jike Song"  
   $ git -config user.email [email]albcamus@gmail.com[/email]  
  
           注意，这样会在当前repository目录下的. git /config中写入配置信息。 如果 git -config加了--global  
        选项，配置信息就会写入到~/. git config文件中。 因为你可能用不同的身份参与不同的项目，而多个  
        项目都用 git 管理，所以建议不用global配置。  
  
$ git -val -l                 //列出 git 变量  
  
   $ git -initdb                 //除非自己创建一个工程，否则没必要使用。 它会创建. git /目录  
                                   //这个目录在 git -clone时也会创建。  
         
   $ git -clone git :// git .kernel.org/pub/scm/linux/kernel/ git /torvalds/linux-2.6. git   
   $ cd linux-2.6  
  
   此时当前目录下有一个. git 目录. 以下我们都在linux-2.6/下演示：  
  
   $ git -pull                 //更新本地的 git tree。 如果自从你clone了linus tree之后，linus tree  
                           //有新的改动，那么把这些更改更新到你的本地tree中  
                        //类似于cvs update  
  
   $ git -diff                 //列出自己本地的tree和linus tree的不同  
                           这也是产生patch的方式。 注意，使用 git -diff产生的patch都应该在patch(1)时指定-p1  
  
   $ git -apply                 相当于patch(1)命令，不过 git -apply专门用来apply那些用 git -diff生成的补丁  
                   --check 不真正打补丁，而只是检查补丁是否能完美的打上  
                -v      verbose  
  
   $ git - gui                  // GUI 模式，不过我觉得q git 最好用  
  
   $ git -rev-list <ID>        以时间为顺序，反向列出revision ID。 也就是先列最新的commit ID。 也可以指定列出的数目，例如：  
                    
   $ git -rev-list -2 971a71bdc9b42e74a5a8ed0433ac27ae92291024  
  
新加文件：  
  
   $ git -add <filename>          //类似于cvs add <filename>  
   $ git -add -a                 //本目录下所有文件和子目录  
   $ git -commit                 //提交。注意需要先add再commit。 又， GIT \_EDITOR环境变量允许你  
                                   //设置编辑器，commit时用它来编辑信息。  
                        -s 选项来为commit message的最后一行添加Signed-off-by: Jike Song <[email]albcamus@gmail.com[/email]>  
  
   $ git -commit -e -s -a  
                           -s 会增加Signed-off-by行，-e会调用vim进行编辑commit message，-a表示all。  
  
删除文件：  
  
   $ git -rm aa.c  
   $ git -commit  
  
恢复删除了的文件：(撤消一次提交)  
  
   $ git -log                  //注意查看上次 git -rm后的 git -commit ID  
   $ git -revert adb2f08089edac8bf1912a618a74485ab42f2b86         //指定导致删除操作的commit ID，进行撤消  
  
列出某一个commit ID对应的补丁：  
  
   $ git -log -1 -p 721151d004dcf01a71b12bb6b893f9160284cf6e  
  
   -1的意思是只显示一个commit。如果你想显示5个，就-5。不指定的话， git log会从该commit一直往后显示。  
  
   还有直接的简洁方法：  
   $ git -format-patch -1 721151d004dcf01a71b12bb6b893f9160284cf6e  
            
        --stdout         //不写入文件，而是打印到标准输出  
  
   或者：  
   $ git -show 721151d004dcf01a71b12bb6b893f9160284cf6e                 //thanks for Xu  
  
   或者：  
   $ git -diff-tree -p 721151d004dcf01a71b12bb6b893f9160284cf6e  
  
  
git -bisect的用法  
/\*{{{\*/  
   $ git -bisect start  
   $ git -bisect bad    //该版本的kernel标记为bad  
  
   或者有针对性的：  
           $ git -bisect bad v2.6.22-rc1  
  
   $ git -bisect good v2.6.22-rc2  
  
  
LABEL:           在你指定了bad和good之后，如果这两个版本之间有1000个revisions， git 就默认剔除了500个，你应该在此时测试该版本：  
            
   创建一个临时性的output目录  
        $ make ../ git \_bisect\_output/  
   编译：  
           $ sudo make O=../ git \_bisect\_output/ menuconfig && make O=../ git \_bisect\_output/ V=1 -j4 && make O=../ git \_bisect\_output/ V=1 modules\_install install  
        注意，最好在menuconfig时，给local version加上一个string，例如step1、step2等。  
  
   启动新编译的kernel，如果还有BUG：  
           $ git -bisect bad  
        goto LABEL; //再编译，验证。  
  
   直到某个时候，只剩下一个revision可以怀疑，那时候就可以确认是它引入了补丁。  
  
   当bisect结束，恢复到master版本：  
            
        $ git -bisect reset  
  
   [注意]  
           git bisect是一个漫长而痛苦的过程。我在Dell Optiplex745(2G内存/Core2双核2G)机器上足足做了一天，才定位到一个commit。  
  
   [replay的用法]  
           如果该输入 git -bisect good的时候，不小心输入了 git -bisect bad，则可以这样：  
                 
                1) git -bisect log | tee ../ git .bisect.log  
                2) 修改../ git .bisect.log，删掉最后两行 -- 也就是回退1步  
                3) git -bisect replay ../ git .bisect.log  
  
   [visualize的用法]  
           git -bisect的时间很长，因为可能需要编译N次内核。 在此期间，可以用：  
                 
                $ git -bisect visualize  
        来在 git k中查看目前还在的那些Revs。  
        FYI: 如果你象我一样更喜欢q git ，可以修改`which git -bisect`脚本，将' git k'字样替换成'q git '。  
   /\*}}}\*/  
                    
            
git -citool GUI 界面的 git -commit。 不但可以提交，而且可以编辑上次commit的信息。  
  
  
git -diff的用法  
  
   列出自己tree和master的不同：  
   $ git -diff  
  
   列出自己的tree和某一个version的不同：  
   $ git -diff v2.6.22  
  
   列出某一个文件，和以前某个version的该文件的不同：  
   $ git diff v2.6.20 init/main.c  
   注意结果中+表示自己的tree，-表示2.6.20的。  
  
   列出两个tag中某一文件的不同：  
   $ git -diff v2.6.23 v2.6.24-rc1 init/main.c  
  
  
  
查看某一文件都被哪些补丁改动过：  
  
   $ git -whatchanged -p security/Kconfig      //thanks to Herbert Xu  
  
  
从本地 git 仓库中提取某个版本的kernel：  
  
   $ git -archive -v v2.6.18 | (cd ../linux-2.6.18/ && tar xf -)  
   -v表示--verbose，注意'v2.6.18'必须是 git -tag -l列出来的tags中的一个。  
  
或者打成tarball:  
  
   $ git -archive -v --format=tar v2.6.24 |bzip2 > ../linux-2.6.24.tar.bz2  
  
  
把当前本地的 git 仓库export到另一个目录(为了测试最新的Linus Tree而又不弄乱 git 仓库)  
  
   $ git -status         //FIXME: 我不确定该用 git -status还是 git -branch  
                                                   //或者别的什么命令  
   $ On branch master  
   nothing to commit (working directory clean)  
   $ git -archive -v master | (cd ../linux-2.6-Jan14/ && tar xf -)  
  
你的本地存储乱了，恢复到mailine：  
  
   $ git -reset --hard                 //WARNING: 这会使你的本地改动全部丢失，慎用！  
  
  
(FIXME: 该方法根本不对！commit log还是乱的！  
                                  或许应该用 git -reset --hard)  
         
        $ git -diff |tee ../local\_against\_mainline.diff  
        $ git -apply -R < ../local\_against\_mainline.diff  
   或者：  
        $ git -diff | git -apply -R  
   即可。  
  
git 分析邮件：  
         
    $ git -mailinfo msg patch <   <your-email-file.eml>   
  
    分析邮件，把commit log写到msg文件，补丁写到patch文件。 其他信息打印到标准输出。  
  
从mail文件中打补丁：  
         
        $ git -am < <your-email-file.eml>  
  
git -send-email:  
  
    例如  
    $ git -send-email --compose --to [email]jeff@garzik.org[/email] --cc [email]davem@davemloft.net[/email] /  
      --cc [email]akpm@linux-foundation.org[/email] --cc [email]netdev@vger.kernel.org[/email] /  
      -cc [email]linux-kernel@vger.kernel.org[/email] --smtp-server <your-smtp-server> /  
      <your-patch-name.patch>  
  
      注意，我的smtp server不需要验证，如果你需要验证，用  
        
              --smtp-user=<your account> --smtp-pass=<your passwd>  
  
       来指定。  
  
  
使用 git -tools来处理mbox里的patch：  
     
        $ git clone git :// git .kernel.org/pub/scm/linux/kernel/ git /torvalds/ git -tools. git   
  
   然后编译、安装。  
   它提供了applypatch, cvs2 git , dotest, mailinfo, mailsplit, stripspace这些工具。  
  
  
email注意事项：  
         
1) 可以用xclip程序来复制补丁  
  
            $ git -format-patch -1 --stdout <SHA-id> | xclip  
      
     然后鼠标中键粘贴。  
  
2) 小心确认你的编辑器  
      
     如使用thunderbird的插件：External Editor指定Vim编辑器等，确认不wrap word。  
     我不太敢确保是否~/.vimrc中设置了：  
              
          set textwidth=0   
          set wrapmargin=0  
  
     就能不自动插入<EOL>。 不过用:r命令读入补丁文件肯定没问题。