



René Vincent Jansen

Customizing Git with NetREXX



An introduction with practical examples of
extending Git with NetREXX

27th International Rexx Language Symposium, Tampa 2016

GIT

- Strong support for non-linear development**
- Distributed development**
- Compatibility with existing systems/protocols**
- Efficient handling of large projects**
- Cryptographic authentication of history**
- Toolkit-based design**
- Pluggable merge strategies**
- Garbage accumulates unless collected**
- Periodic explicit object packing**



Agenda

- ❖ What is Git - What can it do - some background with practical examples
- ❖ Git hooks in NETREXX
- ❖ Git internals exploration in NETREXX

Previously

- ❖ Earlier, there were version management systems called SCCS, RCS, CVS, PVCS, Librarian, Endeavor, Visual SourceSafe etcetera. Some of these are still in use.
- ❖ Some systems are proprietary, closed source systems. Some are quite expensive. This is not always reflected by the feature-set or the ease of use.
- ❖ The open source world has mostly moved from CVS to Subversion (SVN) and from there to Git
- ❖ Other distributed VCS are: Darcs, Mercurial, SVK, Bazaar and Monotone.

What are the advantages of Git

- ❖ Distributed
- ❖ Speed
- ❖ Easy and fast branching and merging
- ❖ Other systems allow branching and merging; Git is designed for it
- ❖ (It is free)

Git history

- ❖ Linux kernel was maintained in a proprietary system called BitKeeper, which had a special provision for free-of-charge use for open source projects
- ❖ The reverse engineering of BitKeepers' protocol (by Andrew Tridgell, to provide a free client) lead to a rift with the copyright holders that left Linux without a version management system overnight.
- ❖ Linus set himself to the task of providing a version management system. He names all his software after himself. (That was, presumably, a joke. We are not sure).

Git design input

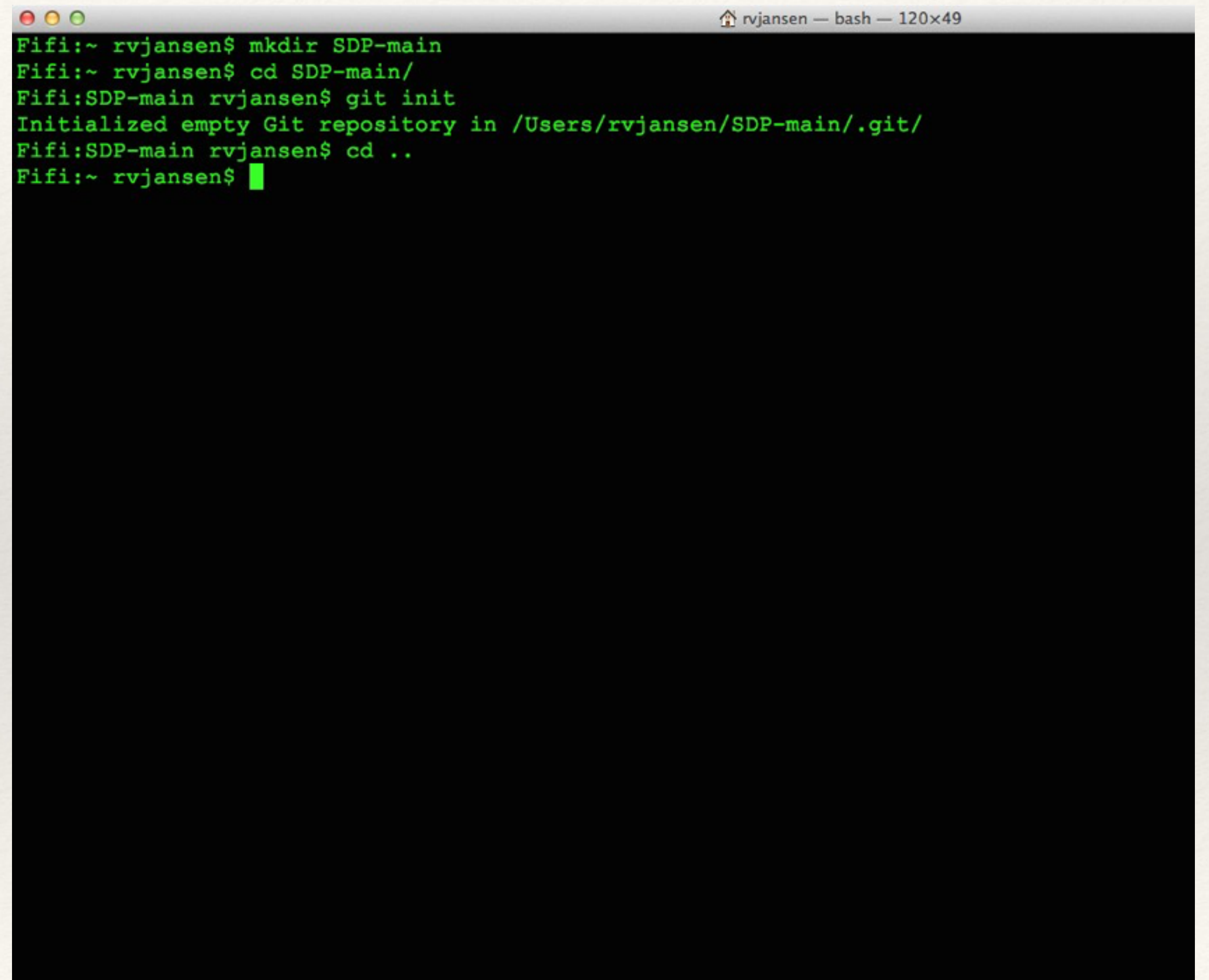
- ❖ Torvald's experience with large scale distributed maintenance of the Linux kernel, and his knowledge of file systems performance led to the following characteristics:
 - ❖ Strong support for non-linear development
 - ❖ Distributed development
 - ❖ Compatibility with existing protocols
 - ❖ Efficient handling of large projects
 - ❖ Cryptographic authentication of project history
 - ❖ Toolkit based design - pluggable merging strategies - no compaction unless needed

Start a repository

- ❖ in any directory, type `git init`
- ❖ this creates a structure in the `.git` subdirectory

Git init

- ❖ the repository data is in the .git subdirectory
- ❖ the rest is called working directory

A terminal window with a dark background and light green text. The window title bar shows a home icon, the name 'rvjansen', and the shell 'bash' with window dimensions '120x49'. The terminal output shows the following commands and their results:

```
Fifi:~ rvjansen$ mkdir SDP-main
Fifi:~ rvjansen$ cd SDP-main/
Fifi:SDP-main rvjansen$ git init
Initialized empty Git repository in /Users/rvjansen/SDP-main/.git/
Fifi:SDP-main rvjansen$ cd ..
Fifi:~ rvjansen$
```

Setting some variables

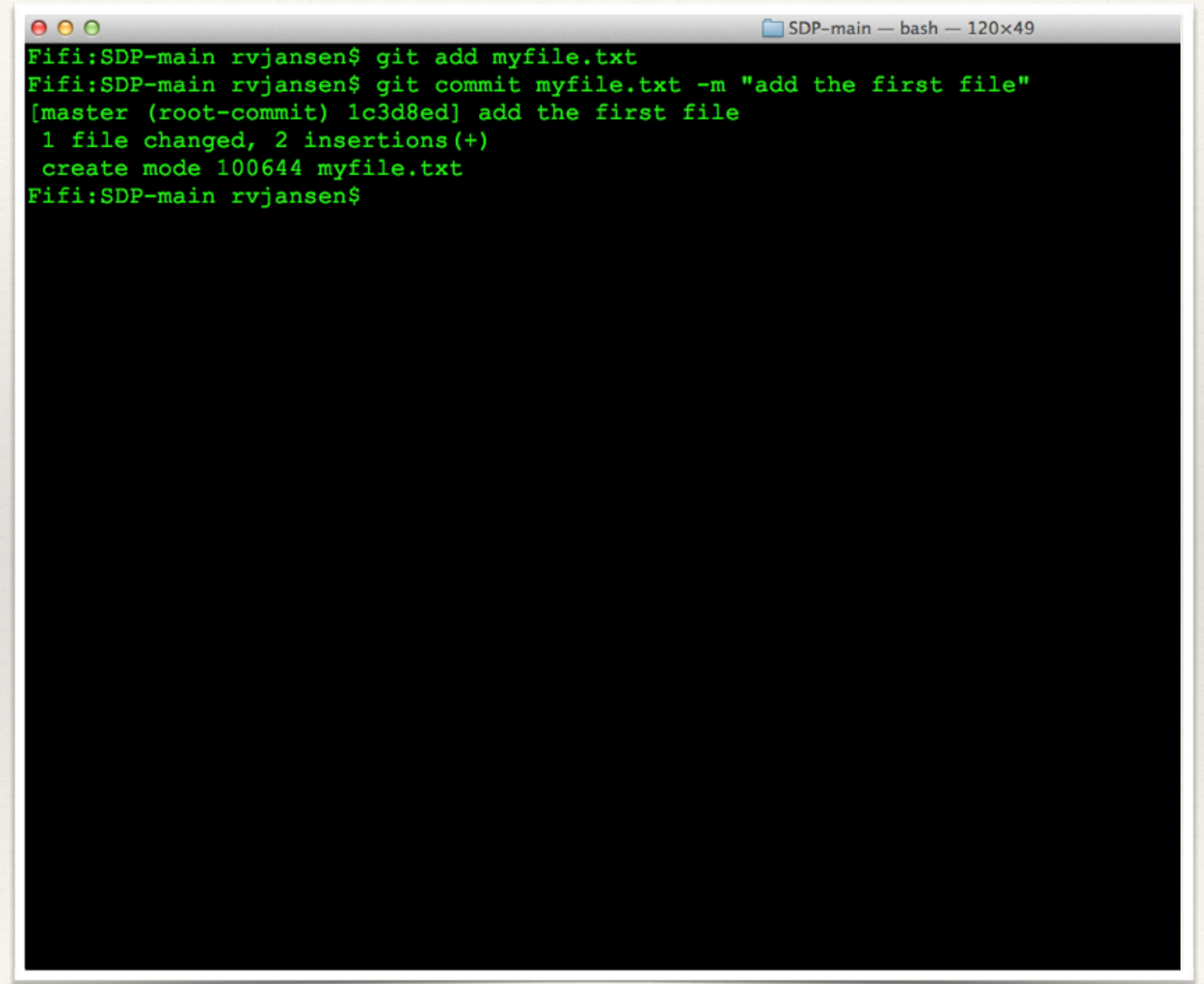
- ❖ The first time you use git, it will ask you for some information
- ❖ It is mainly about your user id and email address
- ❖ You only have to set this once
 - ❖ `git config --global user.name "Rene Vincent Jansen"`
 - ❖ `git config --global user.email rene.jansen@ing.nl`

Adding files, committing them

- ❖ Now you can add files, edit them etc
- ❖ When it is time to make a snapshot of those, “git add” them
 - ❖ `git add mynewfile.txt`
- ❖ After adding them, commit them to the repository
 - ❖ `git commit -m “commit message”`
 - ❖ the commit message is non-optional
- ❖ After this, you can always get back to that version of the file
- ❖ **Shortcut:** `git commit -a -m “automatically commits added files”`

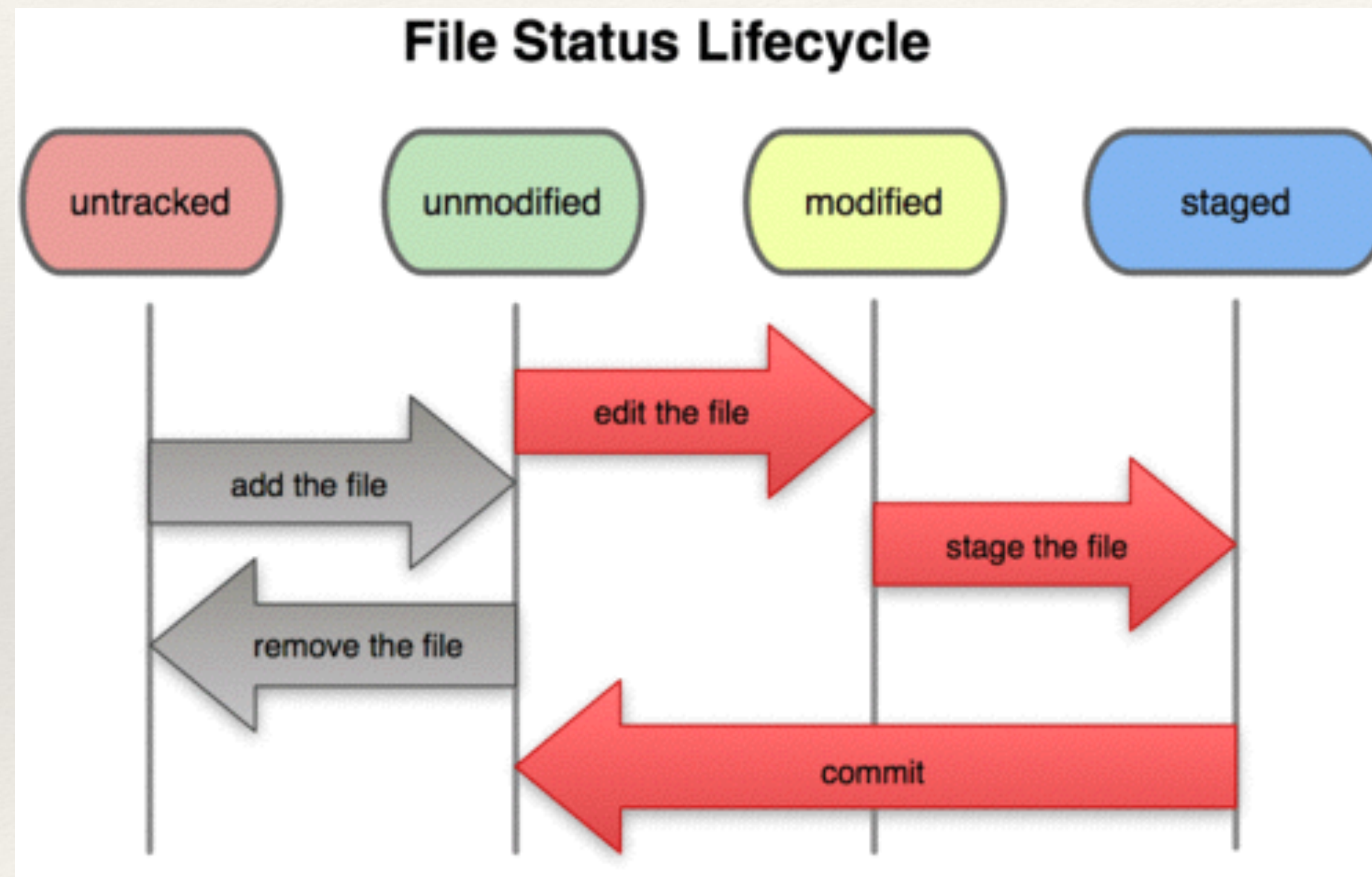
Add and commit a file

- ❖ This commits locally



```
Fifi:SDP-main rvjansen$ git add myfile.txt
Fifi:SDP-main rvjansen$ git commit myfile.txt -m "add the first file"
[master (root-commit) 1c3d8ed] add the first file
 1 file changed, 2 insertions(+)
 create mode 100644 myfile.txt
Fifi:SDP-main rvjansen$
```


File Status Lifecycle



An untracked file needs to be **added**, it can be modified, and needs to be **staged** to be committed. Staging also uses the “`git add`” command.

Use a remote repository

- ❖ This starts with “cloning” a repository
- ❖ For example:
 - ❖ `git clone rvjansen@someserver.com:/SourceREP.git`
 - ❖ this will put a copy of the repository in a subdirectory named SourceREP

Clone a remote repository

```
Fifi:~ rvjansen$ git clone rvjansen@rvjansen.com:SDP-main.git
Cloning into 'SDP-main'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (3/3), done.
Checking connectivity... done.
Fifi:~ rvjansen$ █
```

- ❖ The remote repository is set as origin and default destination for push and pull

ssh

- ❖ git operates standard over the ssh protocol
- ❖ ssh is the secure shell
- ❖ it is included in Linux, macOS standard, and the windows git-bash download
- ❖ it can use other protocols but don't worry about this now

How to get work into a remote repository

- ❖ if you have cloned a repository, you can see what its remote origin is
 - ❖ `git remote -v`
- ❖ after you have committed locally, push your changes to the remote
 - ❖ `git push`
- ❖ if you have pushed them, everybody else can pull them to their local repository

See what you changed/added/deleted

- ❖ `git status`
- ❖ gives a report what changed, what is staged as part of the commit, and which files in your working directory are new - you might have forgotten to add them

See the content of what you changed

- ❖ `git diff`
- ❖ `diff` will show what you changed but not yet staged
- ❖ if you want to see what you staged but not yet committed, type
 - ❖ `git diff - cached`

See status and edits

- ❖ Everything which is changed
- ❖ Tells you to “git add” your changes to add them to the commit

```
Fifi:SDP-main rvjansen$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   myfile.txt

no changes added to commit (use "git add" and/or "git commit -a")
Fifi:SDP-main rvjansen$ git diff
diff --git a/myfile.txt b/myfile.txt
index 77502db..106ebdf 100644
--- a/myfile.txt
+++ b/myfile.txt
@@ -1,2 +1,7 @@
  hello SDP world

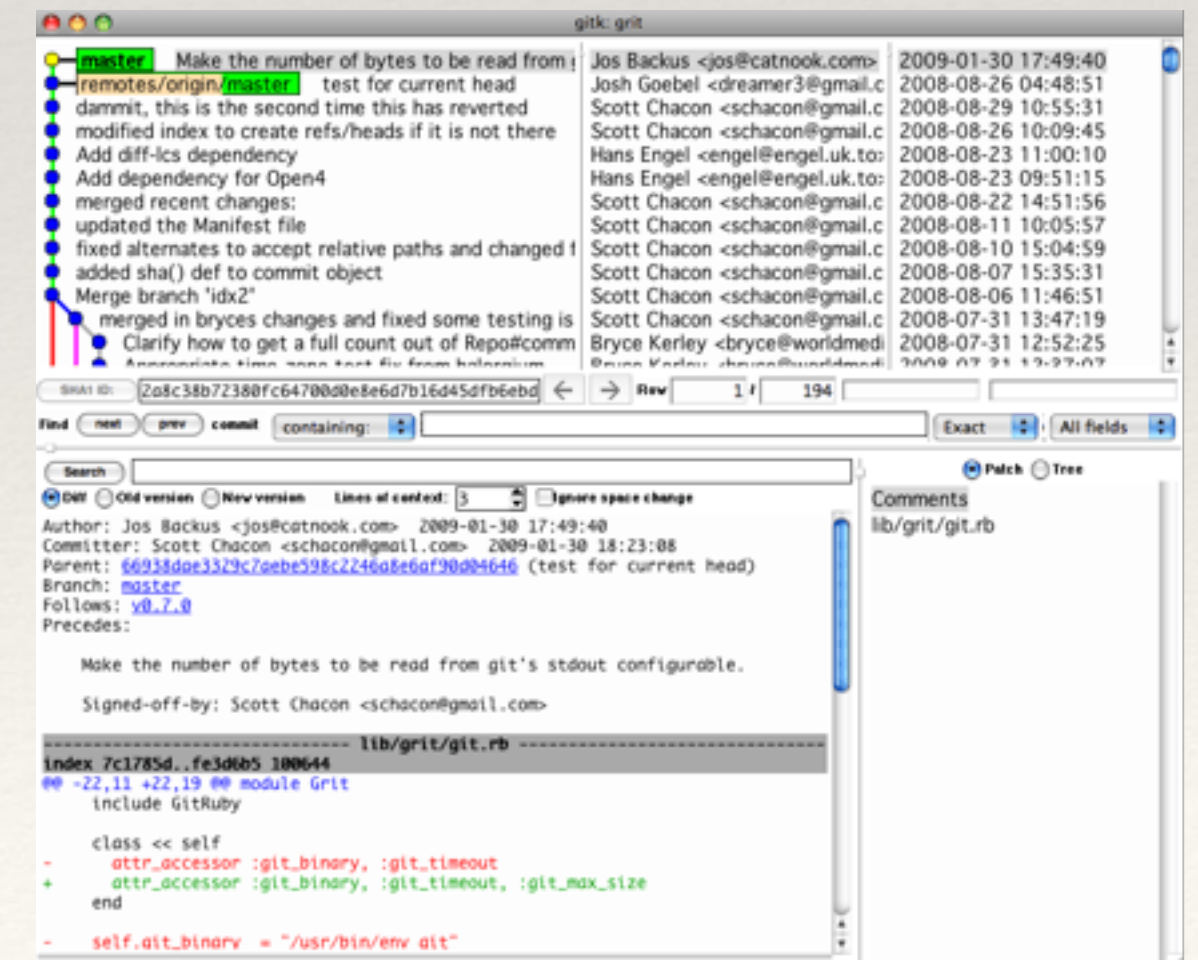
+I added a new line
+
+And after that, another line
+
+
Fifi:SDP-main rvjansen$
```

How to pull changes in

- ❖ After you have once cloned a git repo, you keep pulling in changes by other people (or yourself, from another machine)
- ❖ `git pull`
- ❖ this merges in changes by other people, even in the same file
- ❖ pulling does not touch new files or changes you did not commit
- ❖ if you changed something locally, and did not commit, git asks you to commit (or stash, more about that later) your changes, so nothing gets lost

See who added/changed what

- ❖ `git log`
- ❖ this will show you all the commits for the repository up to the current pull
- ❖ you can use `gitk` or `giblet` to see things gui-like



The screenshot shows the `gitk` graphical interface. The top panel displays a list of commits with their SHA-1 hashes, authors, and dates. The bottom panel shows a diff view for the commit `2a8c38b72380fc647000be86d7b16d45dfb6ebd`, highlighting changes in the `lib/grit/git.rb` file. The diff shows a change in the `attr_accessor` method definition.

```
gitk: grit
2009-01-30 17:49:40 Jos Backus <jos@catnook.com>
2008-08-26 04:48:51 Josh Goebel <dreamer3@gmail.c
2008-08-29 10:55:31 Scott Chacon <schacon@gmail.c
2008-08-26 10:09:45 Scott Chacon <schacon@gmail.c
2008-08-23 11:00:10 Hans Engel <engel@engel.uk.to
2008-08-23 09:51:15 Hans Engel <engel@engel.uk.to
2008-08-22 14:51:56 Scott Chacon <schacon@gmail.c
2008-08-11 10:05:57 Scott Chacon <schacon@gmail.c
2008-08-10 15:04:59 Scott Chacon <schacon@gmail.c
2008-08-07 15:35:31 Scott Chacon <schacon@gmail.c
2008-08-06 11:46:51 Scott Chacon <schacon@gmail.c
2008-07-31 13:47:19 Scott Chacon <schacon@gmail.c
2008-07-31 12:52:25 Bryce Kerley <bryce@worldmedi
2008-07-31 12:37:07 Bryce Kerley <bryce@worldmedi

Make the number of bytes to be read from git's stdout configurable.
Signed-off-by: Scott Chacon <schacon@gmail.com>

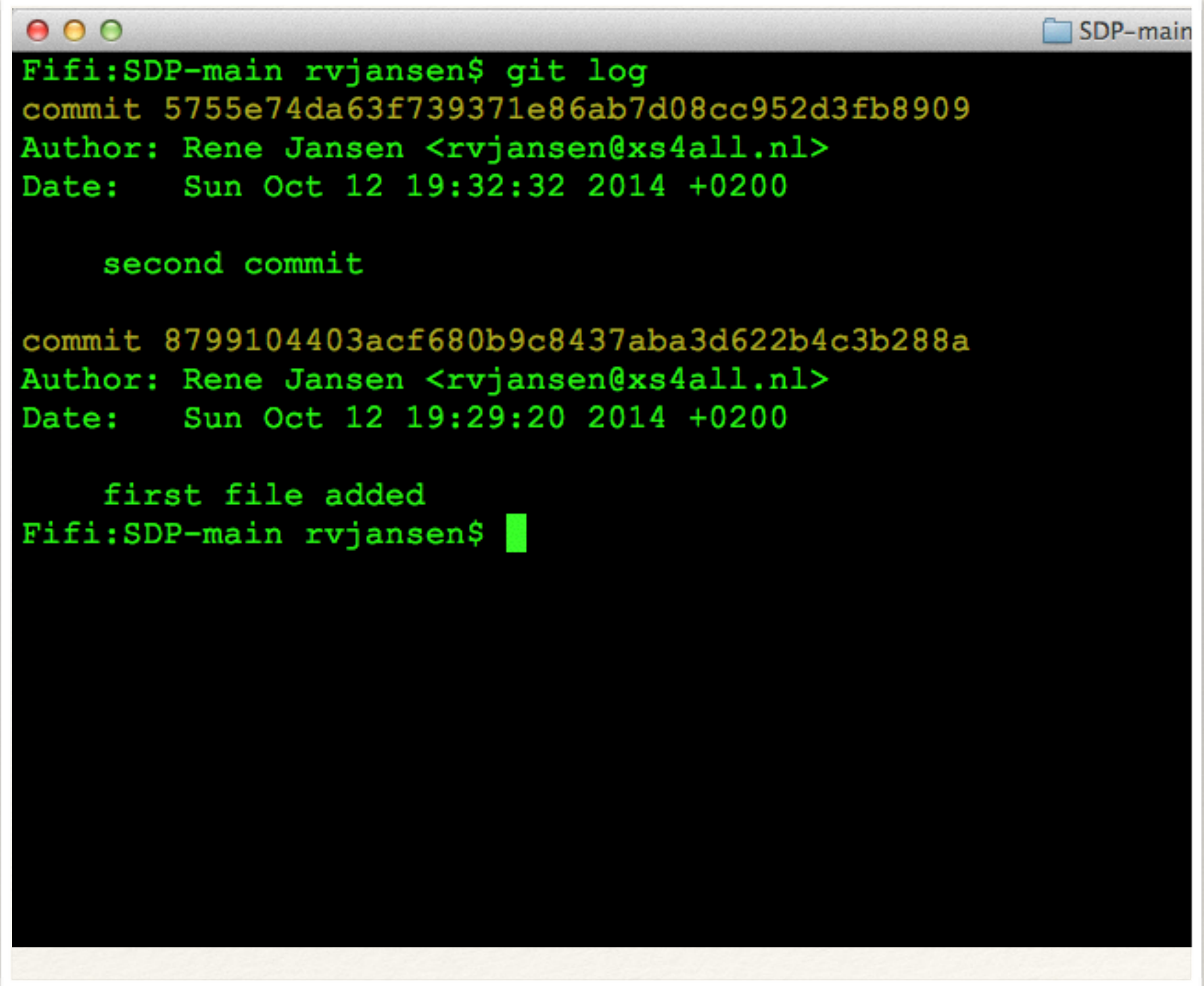
lib/grit/git.rb -----
index 7c1785d..fe3d6b5 100644
@@ -22,11 +22,19 @@ module Grit
  include GitRuby

  class << self
    attr_accessor :git_binary, :git_timeout
+   attr_accessor :git_binary, :git_timeout, :git_max_size
  end

- self.attr_binary = "/usr/bin/env alt"
```


git log

- ❖ There are only two commits in this new repository

A terminal window titled 'SDP-main' showing the output of the 'git log' command. The output lists two commits in reverse chronological order. The first commit (top) has hash 5755e74da63f739371e86ab7d08cc952d3fb8909, author Rene Jansen, and date Sun Oct 12 19:32:32 2014 +0200, with the message 'second commit'. The second commit (bottom) has hash 8799104403acf680b9c8437aba3d622b4c3b288a, author Rene Jansen, and date Sun Oct 12 19:29:20 2014 +0200, with the message 'first file added'. The prompt 'Fifi:SDP-main rvjansen\$' is visible at the end of the output.

```
Fifi:SDP-main rvjansen$ git log
commit 5755e74da63f739371e86ab7d08cc952d3fb8909
Author: Rene Jansen <rvjansen@xs4all.nl>
Date:   Sun Oct 12 19:32:32 2014 +0200

    second commit

commit 8799104403acf680b9c8437aba3d622b4c3b288a
Author: Rene Jansen <rvjansen@xs4all.nl>
Date:   Sun Oct 12 19:29:20 2014 +0200

    first file added
Fifi:SDP-main rvjansen$
```

What is a “version”

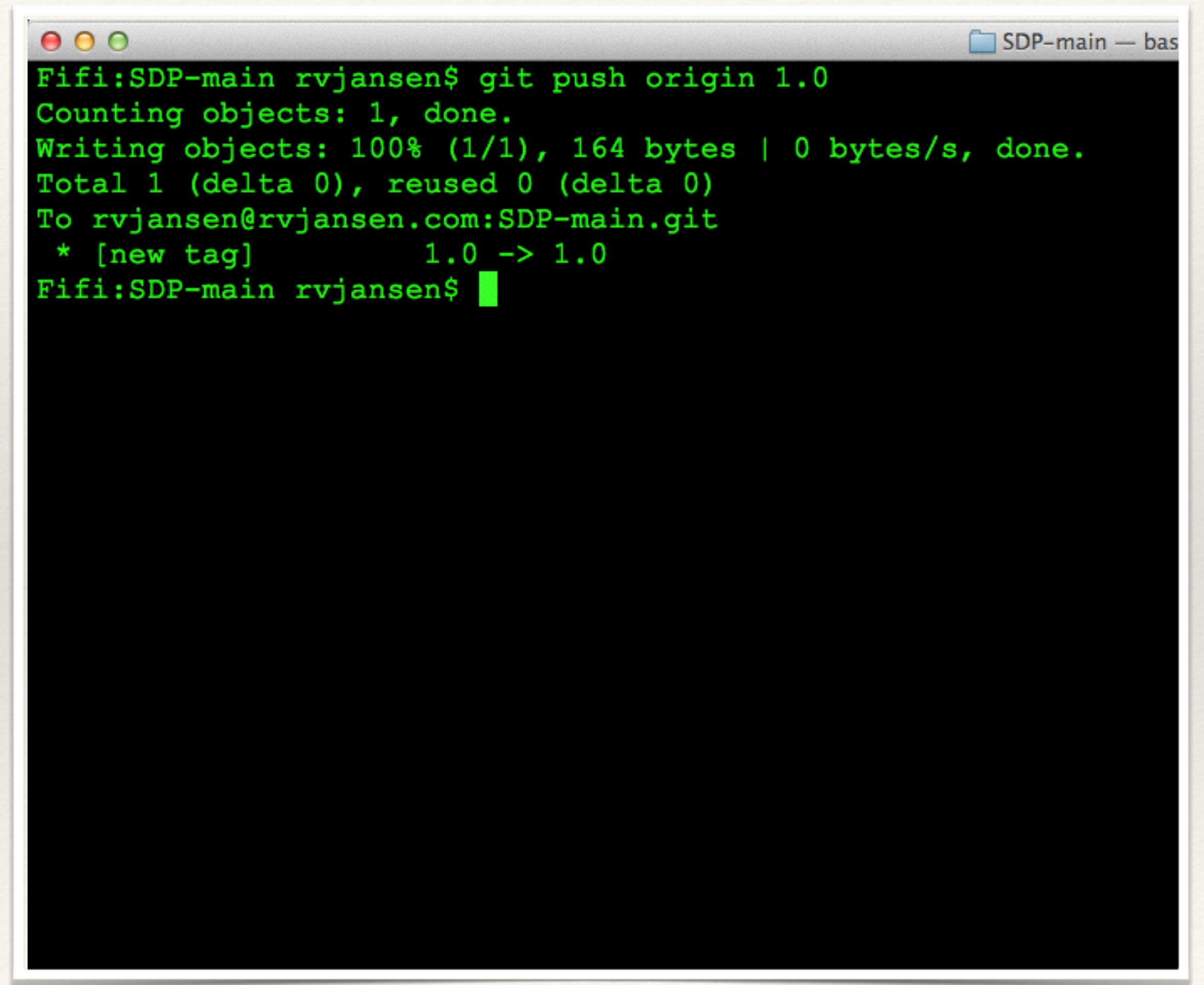
- ❖ A version of a file is represented by a commit, it is an MD5 string
- ❖ Sets of versioned files can be **tagged**, like “release 1.5a”
- ❖ Branches are also named
- ❖ List the tags:
 - ❖ `git tag`

Tagging

- ❖ `git tag -a v1.4 -m "my version 1.4"`
- ❖ tags can be signed with your private key
- ❖ you can also tag later, after the commit already took place
- ❖ tags are not pushed by default to remote repositories
 - ❖ you can run `git push origin [tagname]`

Pushing a tag

- ❖ The local tag will now be part of the remote repository

A terminal window with a black background and green text. The window title is "SDP-main — bas". The prompt is "Fifi:SDP-main rvjansen\$". The command entered is "git push origin 1.0". The output shows the progress of pushing a tag: "Counting objects: 1, done.", "Writing objects: 100% (1/1), 164 bytes | 0 bytes/s, done.", "Total 1 (delta 0), reused 0 (delta 0)", "To rvjansen@rvjansen.com:SDP-main.git", "* [new tag] 1.0 -> 1.0", and the final prompt "Fifi:SDP-main rvjansen\$".

```
Fifi:SDP-main rvjansen$ git push origin 1.0
Counting objects: 1, done.
Writing objects: 100% (1/1), 164 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To rvjansen@rvjansen.com:SDP-main.git
 * [new tag] 1.0 -> 1.0
Fifi:SDP-main rvjansen$
```

How to undo things

- ❖ If you want to discard the changes you made before you staged a file:
 - ❖ `git checkout -- changedfile.txt`
- ❖ It will be overwritten and your changes are lost: this is a **dangerous** command
- ❖ Much better is to stash the file or make a branch - you will save the content

What is a branch?

- ❖ Branching means you can work on the same files, but away from the mainline of development, without disturbing anyone or anything
- ❖ In many version management systems this involves copying the files
- ❖ In git, it does not, and is a fairly cheap and efficient operation
- ❖ This encourages you to branch quickly and often, even in your own repository when trying out alternatives of the code
- ❖ Switching between branches is quick

Branching example

- ❖ Work on a file, let's say a source file and a copybook
- ❖ Create a branch for a story you are working on, let's say a new feature
- ❖ Do some work in that branch
 - ❖ but now ...

There is a production problem ...

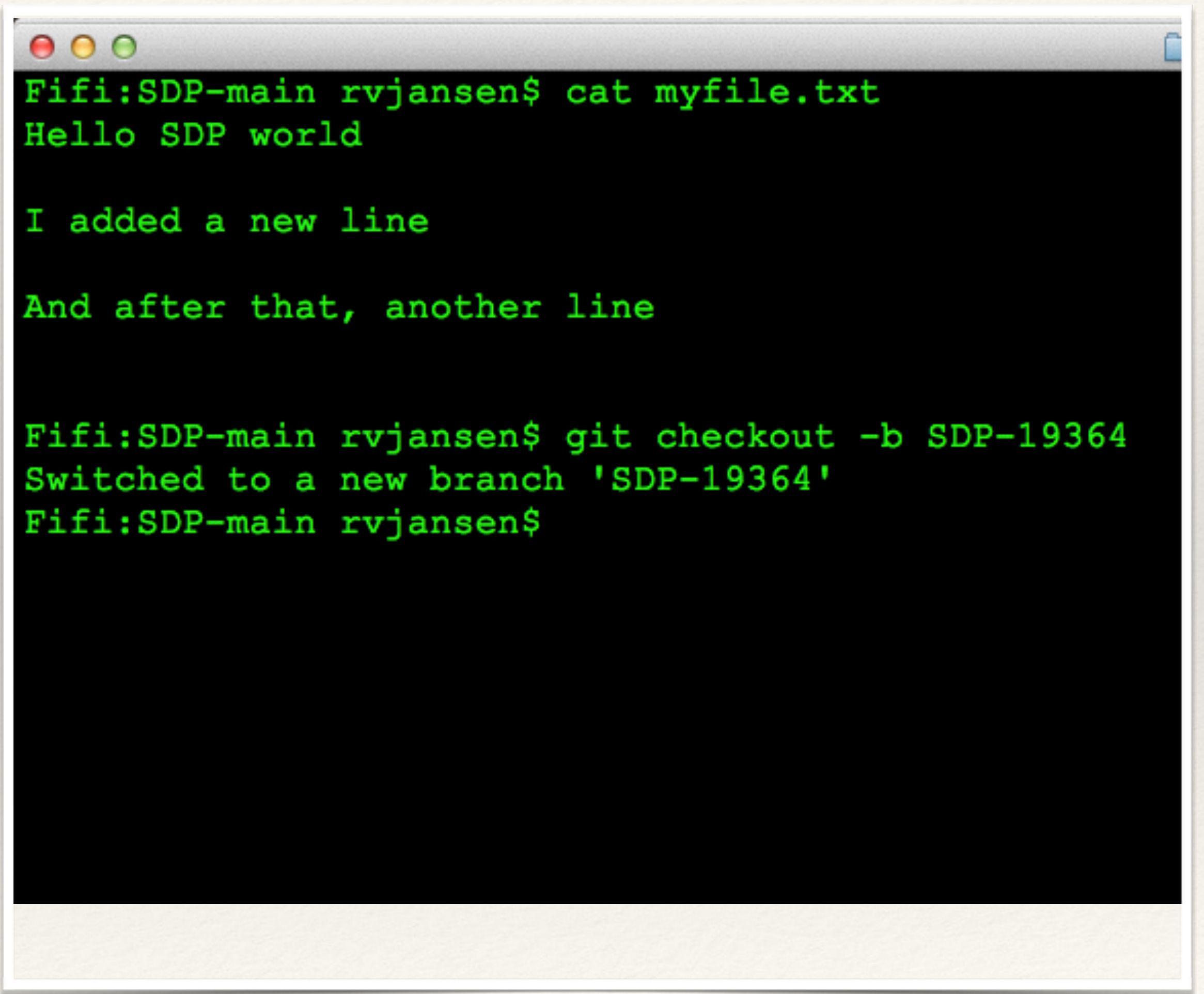
- ❖ Halfway through the work on the new story, there is something that needs fixing in a previous version. There is a persistent deadlock in prod.
- ❖ You now:
 - ❖ Switch back to the production branch
 - ❖ Create a standby branch to create the fix
 - ❖ After it's tested, merge the standby branch, and push to production
 - ❖ Switch back to the original new story branch and keep working

Basic Branching

- ❖ `git checkout -b REX-19364`
- ❖ this is shorthand for
 - ❖ `git branch REX-19364`
 - ❖ `git checkout REX-19364`

Checkout a new branch

- ❖ We are on master (by default)
- ❖ The checkout makes an REX-19364 branch
- ❖ And moves the working directory to it



```
Fifi:SDP-main rvjansen$ cat myfile.txt
Hello SDP world

I added a new line

And after that, another line

Fifi:SDP-main rvjansen$ git checkout -b SDP-19364
Switched to a new branch 'SDP-19364'
Fifi:SDP-main rvjansen$
```

Now work on the branch

- ❖ Edit, edit, commit:
 - ❖ `git commit -a -m "fixed several functional issues, not yet ready"`
- ❖ Now you get the call there is something seriously wrong with the same program in production:
 - ❖ `git checkout master`
- ❖ Now your files are exactly the way before you started working on the new story

Now commit the almost-ready feature branch

- ❖ It was nearly finished, but now there is a production issue
- ❖ Note that I forgot to “git add” the change
- ❖ Then added it to the commit with the -a switch

```
Fifi:SDP-main rvjansen$ cat myfile.txt
Hello SDP world

I added a new line

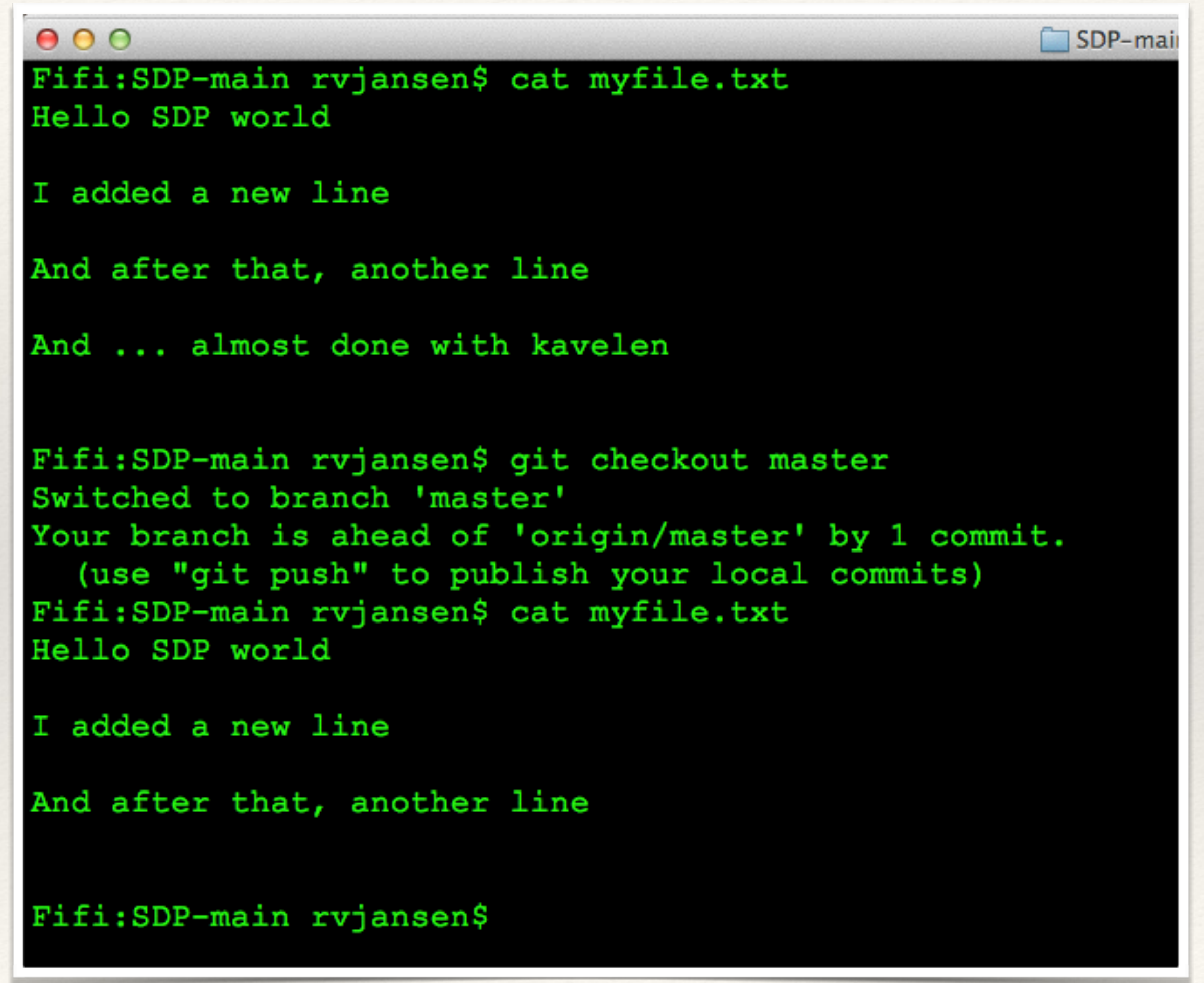
And after that, another line

Fifi:SDP-main rvjansen$ git checkout -b SDP-19364
Switched to a new branch 'SDP-19364'
Fifi:SDP-main rvjansen$ git commit -m "almost fixed kavelen"
On branch SDP-19364
nothing to commit, working directory clean
Fifi:SDP-main rvjansen$ nano myfile.txt
Fifi:SDP-main rvjansen$ git commit -m "almost fixed kavelen"
On branch SDP-19364
Changes not staged for commit:
  modified:   myfile.txt

no changes added to commit
Fifi:SDP-main rvjansen$ git commit -a -m "almost fixed kavelen"
[SDP-19364 5cd27a9] almost fixed kavelen
 1 file changed, 2 insertions(+)
Fifi:SDP-main rvjansen$
```


Checkout the master branch

- ❖ This rolls back the working directory to the previous production state



```
Fifi:SDP-main rvjansen$ cat myfile.txt
Hello SDP world

I added a new line

And after that, another line

And ... almost done with kavelen

Fifi:SDP-main rvjansen$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Fifi:SDP-main rvjansen$ cat myfile.txt
Hello SDP world

I added a new line

And after that, another line

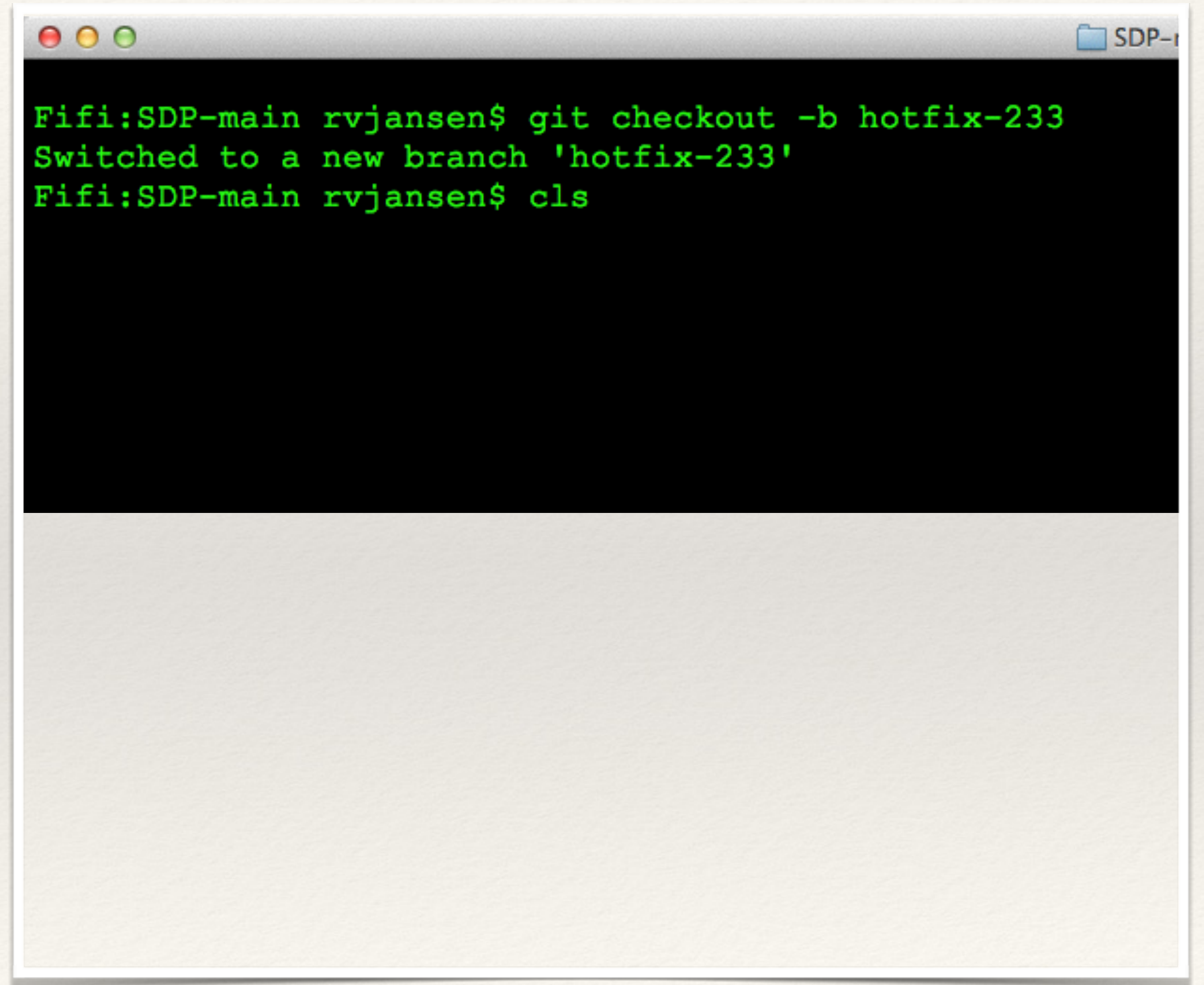
Fifi:SDP-main rvjansen$
```

Now make the standby branch

- ❖ We branch from master, which is production
 - ❖ `git checkout -b hotfix-233`
- ❖ We work on the solution, which is adding some commits to fix a deadlock
 - ❖ `git commit -a -m "fix for prob-233 add commits to rexm123"`
- ❖ Then the fixed branch is sent into the automated testing for Connected-T and after successful functional test (we did not change function, but still) it runs in Connected-A

Checkout hotfix branch

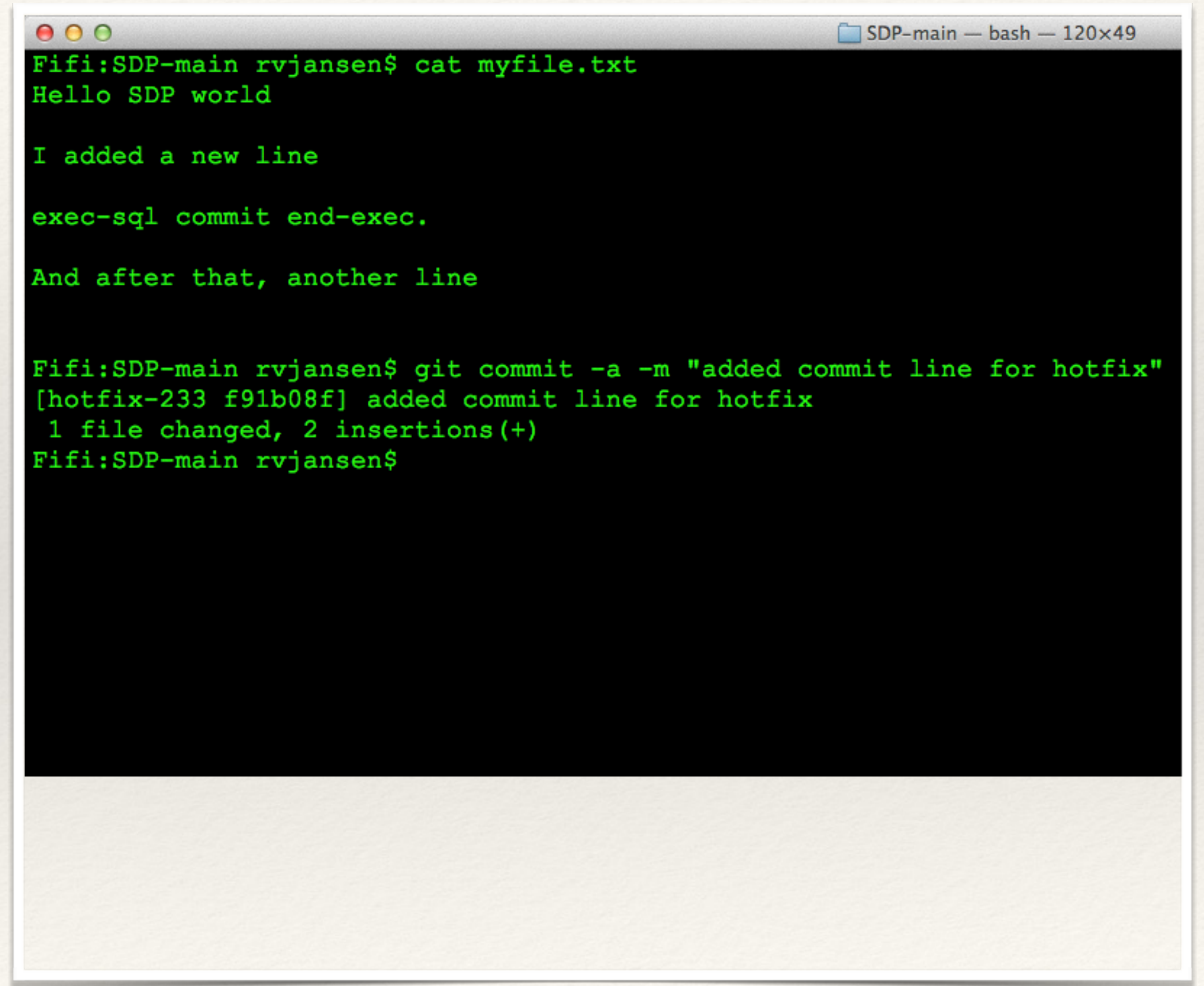
- ❖ this branches from master, which is production

A terminal window with a black background and green text. The window title bar shows three colored circles (red, yellow, green) on the left and a folder icon with the text 'SDP-' on the right. The terminal content shows a user named 'rvjansen' in a directory 'Fifi:SDP-main' running the command 'git checkout -b hotfix-233'. The output shows 'Switched to a new branch 'hotfix-233'' and the user then runs 'cls' to clear the screen.

```
Fifi:SDP-main rvjansen$ git checkout -b hotfix-233
Switched to a new branch 'hotfix-233'
Fifi:SDP-main rvjansen$ cls
```


Now add the fix

- ❖ We need to add a commit

A terminal window titled "SDP-main -- bash -- 120x49" with a dark background and green text. The terminal shows the following sequence of commands and output:

```
Fifi:SDP-main rvjansen$ cat myfile.txt
Hello SDP world

I added a new line

exec-sql commit end-exec.

And after that, another line

Fifi:SDP-main rvjansen$ git commit -a -m "added commit line for hotfix"
[hotfix-233 f91b08f] added commit line for hotfix
 1 file changed, 2 insertions(+)
Fifi:SDP-main rvjansen$
```

Basic Merging

- ❖ Now we have tested the fix, we need to merge the standby branch with master
 - ❖ `git checkout master`
 - ❖ `git merge hotfix-233`
 - ❖ `git branch -d hotfix-233`

Merge and delete the hot fix branch

- ❖ this was auto-merged without much ceremony

```
Fifi:SDP-main rvjansen$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Fifi:SDP-main rvjansen$ git merge hotfix-233
Updating 5755e74..f91b08f
Fast-forward
 myfile.txt | 2 ++
 1 file changed, 2 insertions(+)
Fifi:SDP-main rvjansen$ cat myfile.txt
Hello SDP world

I added a new line

exec-sql commit end-exec.

And after that, another line

Fifi:SDP-main rvjansen$ git branch -d hotfix-233
Deleted branch hotfix-233 (was f91b08f).
Fifi:SDP-main rvjansen$ █
```

Phew

- ❖ Now that we have solved this, let's go back to work and finish the new feature story
 - ❖ `git checkout -b REX-19364`
- ❖ This will reset your working directory to exactly the state it was before we got the call about the deadlock

Resume work on the feature branch

- ❖ This will restore your work on feature REX-19364 to where you left it before the hotfix was needed

```
Fifi:SDP-main rvjansen$ git commit -a -m "merged hotfix"
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
Fifi:SDP-main rvjansen$ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 338 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To rvjansen.com:SDP-main.git
   8799104..f91b08f  master -> master
Fifi:SDP-main rvjansen$ git checkout SDP-19364
Switched to branch 'SDP-19364'
Fifi:SDP-main rvjansen$ cat myfile.txt
Hello SDP world

I added a new line

And after that, another line

And ... almost done with kavelen

Fifi:SDP-main rvjansen$ █
```


Merge feature branch into master

- ❖ when it is ready to be tagged and released
- ❖ this also merged cleanly

```
Fifi:SDP-main rvjansen$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
Fifi:SDP-main rvjansen$ git pull
Already up-to-date.
Fifi:SDP-main rvjansen$ cat myfile.txt
Hello SDP world

I added a new line

exec-sql commit end-exec.

And after that, another line

Fifi:SDP-main rvjansen$ git merge SDP-19364
Updating f91b08f..94e7358
Fast-forward
 myfile.txt | 2 ++
 1 file changed, 2 insertions(+)
Fifi:SDP-main rvjansen$ cat myfile.txt
Hello SDP world

I added a new line

exec-sql commit end-exec.

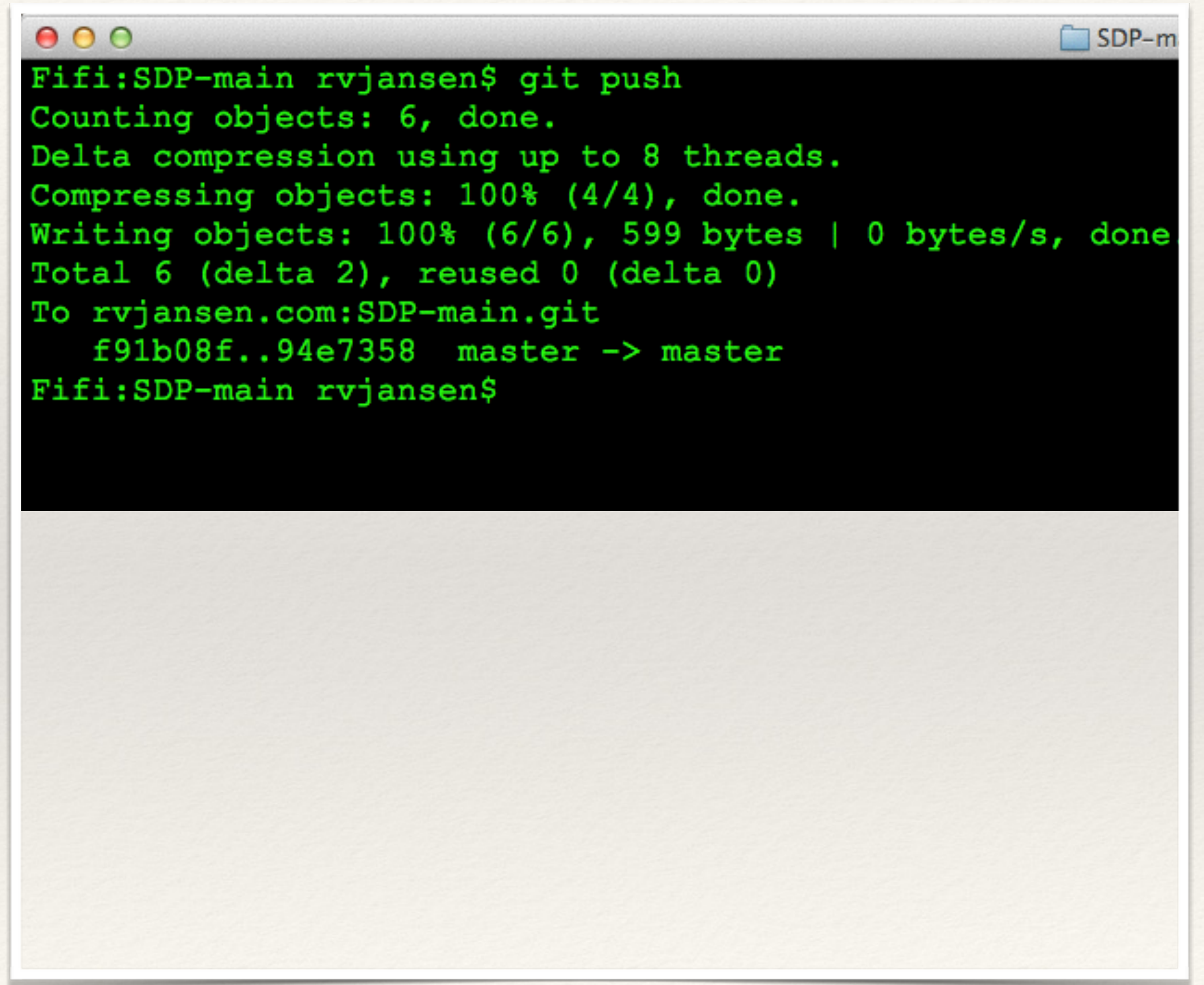
And after that, another line

And ... almost done with kavelen

Fifi:SDP-main rvjansen$
```


Now push everything to origin

- ❖ Now every team member can merge these changes by issuing a “git pull”

A terminal window with a black background and green text. The window title bar shows three colored circles (red, yellow, green) on the left and a folder icon labeled 'SDP-m' on the right. The terminal output shows the execution of 'git push' and its progress: counting 6 objects, compressing them with up to 8 threads, and writing them to the remote repository. The final output shows the commit hash 'f91b08f..94e7358' being pushed to the 'master' branch.

```
Fifi:SDP-main rvjansen$ git push
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 599 bytes | 0 bytes/s, done
Total 6 (delta 2), reused 0 (delta 0)
To rvjansen.com:SDP-main.git
    f91b08f..94e7358  master -> master
Fifi:SDP-main rvjansen$
```

Sometimes, there are merging conflicts

- ❖ Maybe another colleague started working on this, because another manager called him. You both modified a part of the same source file
- ❖ This is no big problem, git tells you there is a conflict, and where
- ❖ This means editing the file that has the difference listed in it, and committing once more, until the files merge cleanly

Server Side Git

- ❖ To put a shared repository on a server, you need a **bare** repository
- ❖ A bare repository does not have a working directory
- ❖ `git clone —bare name-of-repository`
- ❖ `scp -r bare-repository.git userid@server.com:`



GIT Hooks in NETREXX

Customize GIT Behaviour

GIT Hooks

- ❖ a hook is what we used to call an exit
- ❖ in the `.git/hooks` directory there are various samples
- ❖ we will focus on the pre-commit exit
- ❖ this exit gets control when we are committing locally
- ❖ to have it running, rename `pre-commit.sample` to `pre-commit`

Two Examples

- ❖ We will show 2 examples of a pre-commit exit
 - ❖ One will add locking behaviour to GIT
 - ❖ The second will auto-convert UTF16-BE to UTF8

Lockhook part I

```
**
* class lockHook is a git pre-commit hook. It needs to be called from a hook called pre-commit
* situated in the .git/hooks subdirectory. Suggested content of this hook is:
* #!/bin/sh
* java -cp "lib\NetRexxF.jar;bin" lockHook
* when this script returns 0, the commit can go through
* when it returns 1, the commit process is stopped.
*/
options binary nodecimal
import org.netrexx.address.

class lockHook

    properties private static
    fileList = ArrayList()

    method lockHook()
        lockFile = BufferedReader(FileReader("bin/LockedFiles.txt")) -- input file
        loop forever
            line = Rexx lockfile.readLine()
            if line = null then leave
            fileList.add(line)
        end
```



```
/**  
 * Method main asks git for the changed files for this commit.  
 * @param args is a String[]  
 */
```

```
method main(args=String[]) static  
  l = lockHook()  
  /* check names of file to commit */  
  say 'Starting lock check commit hook'  
  command = ArrayList()  
  command.add('git')  
  command.add('diff-index')  
  command.add('--name-only')  
  command.add('--cached')  
  command.add('HEAD')  
  os = OSProcess()  
  a = os.outtrap(command)  
  i = a.iterator()  
  loop while i.hasNext()  
    filename = Rext i.next()  
    say 'lockHook checked:' filename '\-'  
    if l.checkIfLocked(filename) then  
      do  
        say 'locked. Commit aborted.'  
        exit 1  
      end  
    else  
      do  
        say 'not locked.'  
      end  
  end -- loop while i  
  
  exit 0  
  
method checkIfLocked(filename)  
  if fileList.contains(filename) then return 1  
  else return 0
```


checkUTF I

```
**
* class checkUTF is a git pre-commit hook. It needs to be called from a hook called pre-commit
* situated in the .git/hooks subdirectory. Suggested content of this hook is:
* #!/bin/sh
* java -cp "lib\NetRexxF.jar;bin" checkUTF
* when this script returns 0, the commit can go through
* when it returns 1, the commit process is stopped.
* This class is designed to always let the commit go through,
* but keep UTF-16LE files out of the repository.
* It is, for the moment, dependent on class OSProcess.
*/
class checkUTF
/**
 * Method main asks git for the changed files for this commit. For each file:
 * it checks if the file is UTF-16LE, by calling method testByteOrderMark
 * if UTF-16LE, it calls convertToUTF8
 * @param args is a String[]
 */
method main(args=String[]) static
/* check characters sets of files to commit */
command = ArrayList()
command.add('git')
command.add('diff-index')
command.add('--name-only')
command.add('--cached')
command.add('HEAD')
os = OSProcess()
a = os.outtrap(command)
i = a.iterator()
loop while i.hasNext()
    filename = Rexx i.next()
    encoding = testByteOrderMark(filename)
    if encoding = null then do
        exit 0
    end
    if encoding = 'UTF-16LE' then do
        say filename 'is' encoding ' - converting to UTF-8'
        convertToUTF8(filename)
    end
end
exit 0
```



```
/**
 * Method testByteOrderMark tests for a filename supplied as argument if
 * the first two bytes are FF FE, the BOM for UTF-16LE
 * It returns the result to the calling method
 * @param filename is a Rexx
 */
method testByteOrderMark(filename) static signals IOException
/* check for UTF16LE BOM (byte-order-mark) FF FE */
do
  source = DataInputStream(FileInputStream(filename))
  ch1     = Rexx source.readUnsignedByte()    -- read one byte
  ch2     = Rexx source.readUnsignedByte()    -- read one byte
  source.close()
  if ch1.d2x(2) = 'FF' & ch2.d2x(2) = 'FE' then do
    return 'UTF-16LE'
  end
  return 'not UTF-16LE'
catch FileNotFoundException
  say 'file' filename 'not found. continuing with commit.'
  return null
end

/**
 * Method convertToUTF8 takes a filename as parameter and shells iconv to convert to UTF-8
 * After this, the resulting UTF-8 file is renamed to the original name
 * We are not using JVM's rename, because this intermittently fails on windows
 * Because of this, we shell a mv command. This is available due to the git bash shell we run.
 * The file is re-added to the git commit, so that the repository is UTF-8 clean.
 * @param filename is a Rexx
 */
method convertToUTF8(filename) static signals IOException
newfilename = filename||'.utf8'
outFile = FileWriter(newfilename)
dest     = PrintWriter(outFile)
command  = ArrayList()
command.add('iconv')
command.add('-f')
command.add('UTF-16LE')
command.add('-t')
```



```
command.add('UTF-8')
command.add(filename.toString())
os = OSProcess()
a = os.outtrap(command)
i = a.iterator()
loop while i.hasNext()
  line = Rexx i.next()
  dest.println(line)
end
dest.close()
```

```
command = ArrayList()
command.add('mv')
command.add(newfilename.toString())
command.add(filename.toString())
os = OSProcess()
a = os.outtrap(command)
i = a.iterator()
loop while i.hasNext()
  line = Rexx i.next()
  dest.println(line)
end
```

-- and add it again to not commit the UTF-16LE version

```
command = ArrayList()
command.add('git')
command.add('add')
command.add(filename.toString())
os = OSProcess()
a = os.outtrap(command)
i = a.iterator()
loop while i.hasNext()
  line = Rexx i.next()
  dest.println(line)
end
```




Explore and Control

Git Internals in NETREXX

using jGit

So how does it do it?

JGIT

- ▶ Git has implementations in C(++) and Java
- ▶ JGit is the JVM version that we used
- ▶ Build it from source; only Java SDK needed
 - ▶ Maven build, dependencies loaded
 - ▶ Pre-built integrated .sh script no good for z/OS
 - ▶ need jgit-cli.jar

SOME ASSEMBLY REQUIRED

- ▶ Java is UTF16 internally - you would think it will all work
- ▶ currently some 'optimization' has wrong assumptions
 - ▶ and more specific, assumes the world is ASCII (ISO8859-1)
- ▶ This is mostly centered around the display of git diff
- ▶ Git diff and merge work fine
- ▶ Git hooks work fine

RETRIEVE A FILE FROM A GIT REPOSITORY IN AN APPLICATION

```
/**
 * Method retrieveFileFromSHA returns a file by its SHA
 * @param oid is a ObjectId which must be initialized with the SHA
 * @param fileName represents the output filename
 */
method retrieveFileFromSHA(oid=ObjectId, fileName) protect
  init()
  objectReader_ = this.repository_.newObjectReader()
  objectLoader_ = objectReader_.open(oid)

  fops = BufferedOutputStream(FileOutputStream(fileName))
  do
    objectLoader_.copyTo(fops)
  catch MissingObjectException
    say 'missing object'
  catch IOException
    say 'error in i/o'
  end
  fops.close()
close()
```


USE THE PLATFORM DIFF (WHICH C-GIT ALSO SOMETIMES DOES)

```
/**
 * Method diff calls the native diff utility to compare two files
 * which might be EBCDIC encoded and still will be treated as text
 * @param sha1 is a ObjectId
 * @param sha2 is a ObjectId
 */
method diff(sha1=ObjectId,sha2=ObjectId) protect
    retrieveFileFromSHA(sha1,'tmpf1')
    retrieveFileFromSHA(sha2,'tmpf2')

    command      = ArrayList()
    command.add('diff')
    command.add('tmpf1')
    command.add('tmpf2')
    os = OSProcess()
    a = os.outtrap(command)
    i = a.iterator()
    loop while i.hasNext()
        line      = Rextx i.next()
        say line
        if line.pos('is binary') >0 then cmp(sha1,sha2)
    end
```


USE THE NATIVE CMP FOR BINARY FILES (WHICH GIT DOESN'T DO)

```
/**  
 * Method cmp compares two binary files  
 * @param sha1 is a ObjectId  
 * @param sha2 is a ObjectId  
 */  
method cmp(sha1=ObjectId,sha2=ObjectId) protect  
    retrieveFileFromSHA(sha1,'tmpf1')  
    retrieveFileFromSHA(sha2,'tmpf2')
```

```
command    = ArrayList()  
command.add('cmp')  
command.add('tmpf1')  
command.add('tmpf2')  
os = OSProcess()  
a = os.outtrap(command)  
i = a.iterator()  
loop while i.hasNext()  
    line = Rexx i.next()  
    say line  
end
```


COPY HFS FILES FROM/TO SAM (QSAM OR PDS(/E))

```
/**
 * Method copy shells the cp command to copy a file
 * one of those might be QSAM, following the z/OS convention
 * example: copy('//''M06F947.CHARS.TEST''','bla.tst')
 * Does not use the repository, but enables to add files to the workspace
 * @param fileNameTo is a Rexx
 */
method copy(fileNameFrom,fileNameTo) protect
  command      = ArrayList()
  command.add(String('cp'))
  command.add(String(fileNameFrom))
  command.add(String(fileNameTo))
  os = OSProcess()
  a  = os.outtrap(command)
  i  = a.iterator()
  loop while i.hasNext()
    line      = Rexx i.next()
    say line
end
```


WALKING COMMITS

```
/**
 * Method commitWalk ...
 * @param head is a ObjectId
 * @return TreeMap containing ...
 */
method commitWalk(head=ObjectId,quiet) returns TreeMap protect
init()
this.filesMap.clear()
rev = RevCommit
do
  walk = RevWalk(this.repository_)
  commit = walk.parseCommit(head)
  walk.markStart(commit)
  count = int 0
  loop forever
rev = walk.next()
if rev = null then leave
say '-'.copies(40)
say rev
say 'Commit Time   :' Date(long rev.getCommitTime() * 1000)
say 'Author ID     :' rev.getAuthorIdent
say 'Committer ID  :' rev.getCommitterIdent
say 'Commit Message:' rev.getFullMessage()

say 'tree: (Files in Commit Object)'
treeWalk(rev.getTree(),quiet)
say
count = count + 1
end
say count "commits in this branch"
walk.dispose()
close()
end
return this.filesMap
```


WALK A FILES TREE

```
/**
 * Method treeWalk ...
 * @param treeId is a ObjectId
 */
method treeWalk(treeId=ObjectId,quiet=0) protect
  init()
  count = 0
  treeWalk_ = TreeWalk(this.repository_)
  treeWalk_.addTree(treeId)
  treeWalk_.setRecursive(1)
  treeWalk_.setPostOrderTraversal(1)
  loop while treeWalk_.next()
    fileMode = Integer.parseInt(treeWalk_.getFileMode(0).toString() )
    objectId_ = treeWalk_.getObjectId(0).name()
    path = treeWalk_.getPathString()
    count = count + 1
    if \quiet then say objectId_ (Rexx fileMode).format(6) path
    if this.filesMap.get(path) = null then
do
  this.filesMap.put(path,objectId_)
end
  else
do
  objids = this.filesMap.get(path)
  objids = objids Rexx(objectId_.toString())
  this.filesMap.put(path,objids)
end
  end
  if \quiet then say count "files in commit tree"
  close()
```

Question?

- ❖ Thanks for your attention!
- ❖ rvjansen@xs4all.nl