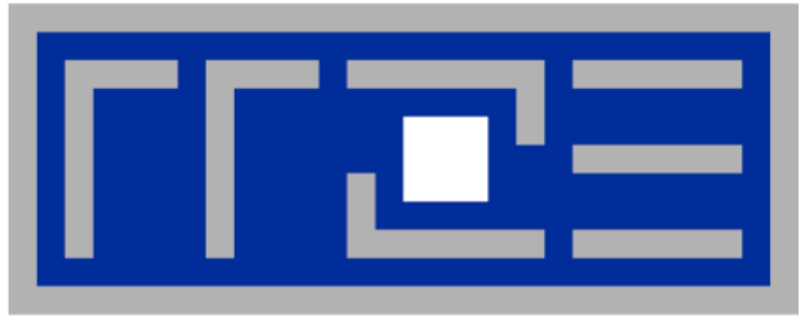


Introduction to version control with SVN and GIT

(on UNIX)

rev.1

Michele MARTONE

LRZ
Garching bei Muenchen, Germany

March 1, 2019

Why version control ?

File names and contents evolve over time

```
thesis.tex
thesis_prof.tex
thesis_prof_rev1.tex
thesis_prof_rev1_commented.tex
thesis_prof_rev1_commented2.tex
thesis_prof_rev1_commented3.tex
thesis_prof_rev1_corrected.tex
thesis_newchap.tex
thesis_final.tex
thesis_final_1_june.tex
thesis_final_3_june.tex
thesis_FINAL_PRE.tex
thesis_FINAL.tex
thesis_FINAL_.tex
thesis_FINAL__.tex
```

Why version control ?

Still asking why ?

- keep track of file changes over time
- work from many systems
- collaborate
- distribute to a wider audience (publish)
- understand whom to **blame** for a bug !
- ...

Evolution

1972–

`sccs`	*deltas*
`rcs`	*speed, robustness*
`cvs`	*remote operation*
`svn`	*atomic transactions, any file, attributes*
`hg, bzr, darcs`	*distributed, user friendly*
`git`	*distributed, flexible*

This presentation is...

- limited to the basics of `svn` and `git`
- skipping much of the theory
- rich in ready examples (from page 64)

During this presentation...

Ask! Especially if...

- I give too much for granted
- a concept is not clear

After this session you...

- should have a **basic** command of `svn` and `git`
- can deepen theory: see references at page 94
- are invited to take slides home and use extra examples (page 80)

Notation and all those coloured boxes

green box

example or description

```
1 echo 'shell commands and output' # comment
```

```
1 shell commands and output
```

blue box

example or description

- `svn #svn-centric shell commands coloring`
- `git #git-centric shell commands coloring`
- *generic command output (stdout) coloring*

red box

Caution required!

Contents overview

Intro
Glossary
Locally
SVN
GIT

Collaboratively
SVN
GIT

Exercises
SVN
GIT
Extra
SVN+GIT
Outro
References

Not all bytes are ASCII

First bit of ASCII byte is unset

00000000 00000001 ... 01111111: ASCII
10000000 10000001 ... 11111111: ISO 8859-1 or other

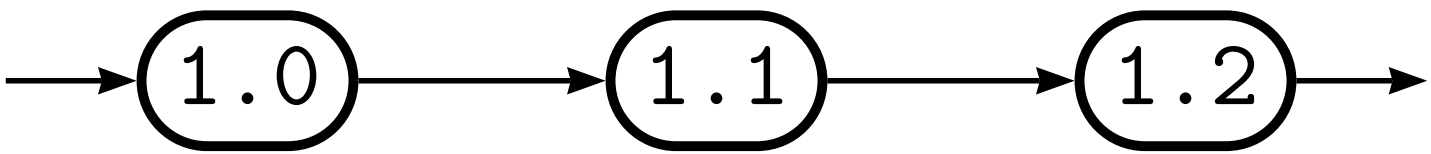
Example[1]

"o" (*01101111*) is ASCII[2]
"ö" (*11000011 10110110*) is not[3]

[1] See also extra example at page 78.
[2] Extra: see `man ascii`.
[3] Extra: see `man charsets`.

Major public milestones

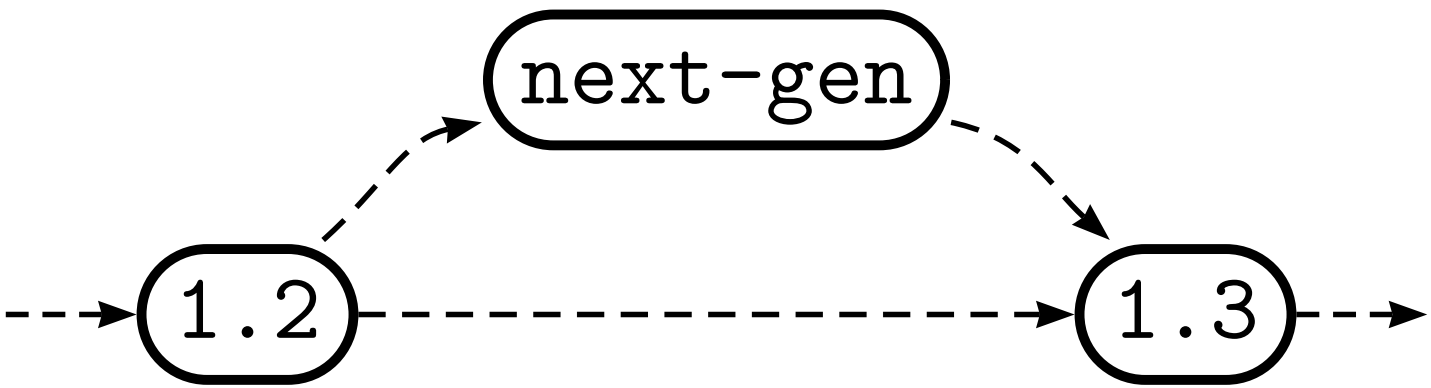
I develop software: `foo`.
On a periodic basis, I publish new versions (**releases**): `foo-1.0`, `foo-1.1`, ...



Toward a milestone

I have a disrupting idea how to improve `foo`.
But I need to work a bit on it.

I start developing a separate **version**.
E.g.: `foo-next-gen`.
When ready, I release it (mailing lists, etc., ...).

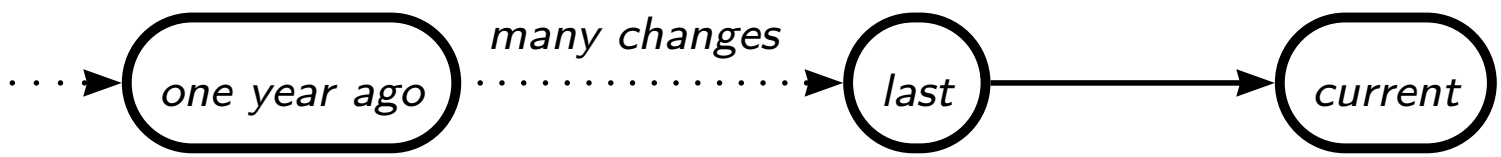


Evolving over time

I develop intensely a large program over a long time.
Once stable, I just add feature every while.
After a year, I discover a bug on some rarely occurring input.

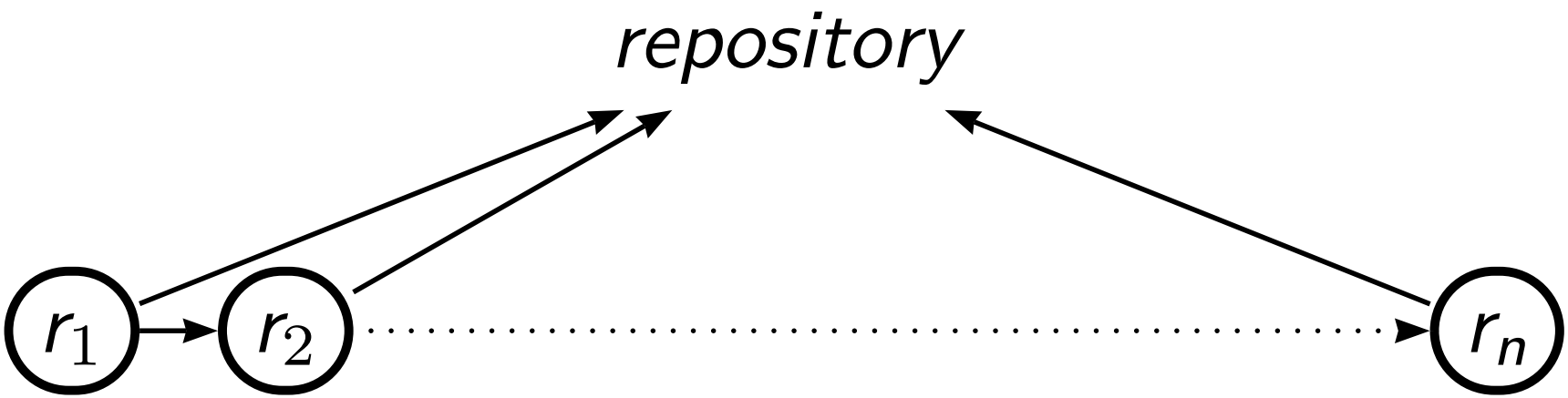
Did I break it on the way ?

Do I have the **revisions** history to check ?



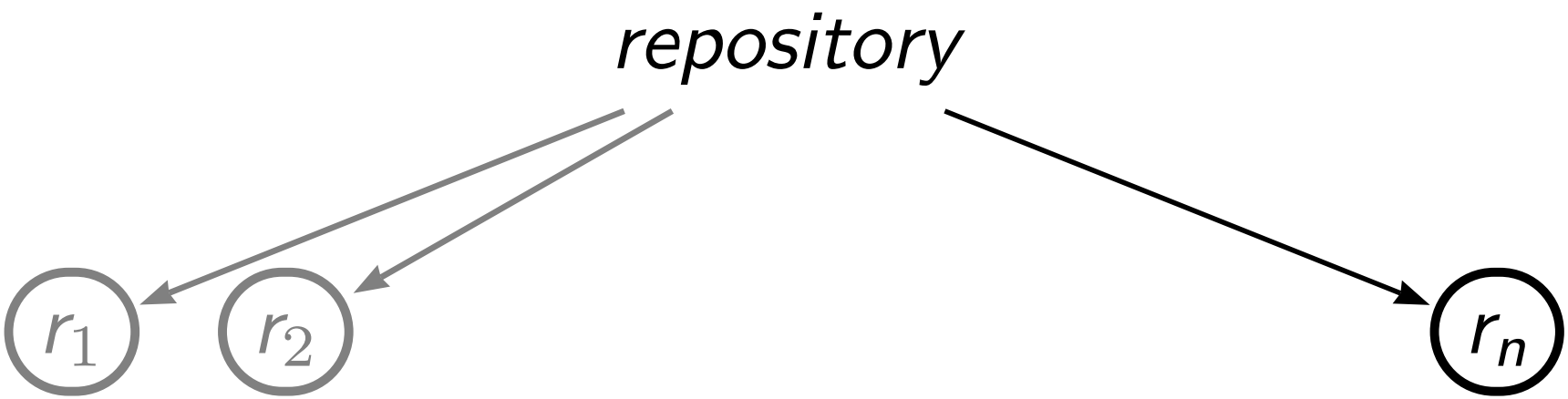
Keeping revisions snapshots

I saved my past revisions in a **repository**.
A revision can be referred by a code.
In the simplest case, sequential (e.g. an integer).



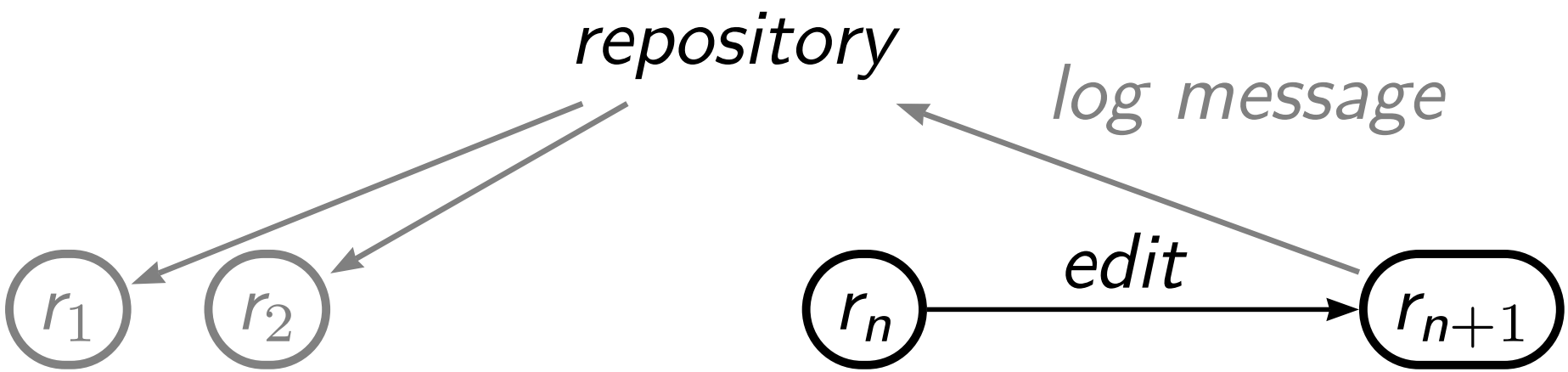
Get snapshots to work with

I request and get a particular revision of my files.
This is my **working copy**.



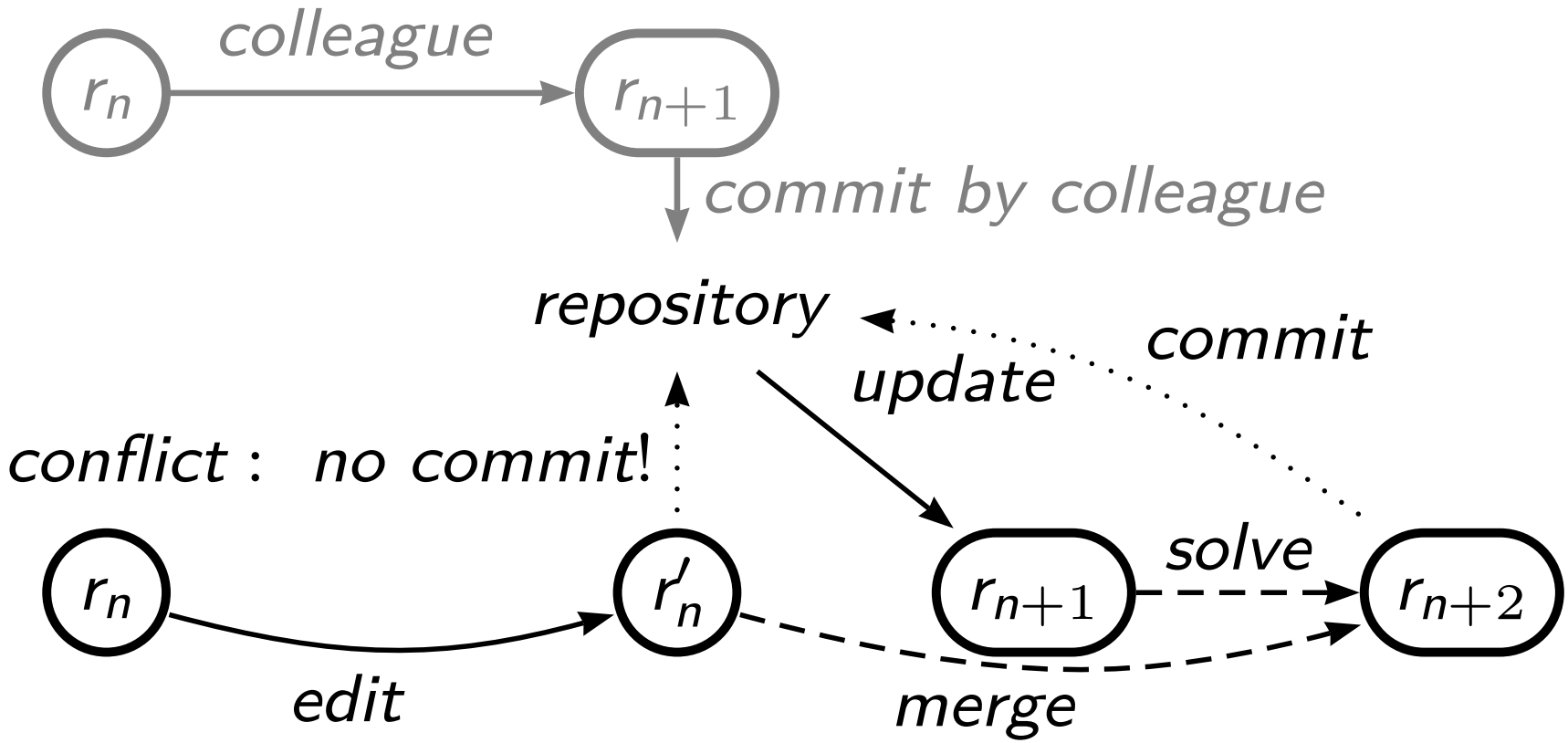
My modified snapshot

I change the code.
Ready to **commit** my changes to new revision, and give a comment.
This comment is called **log message**.



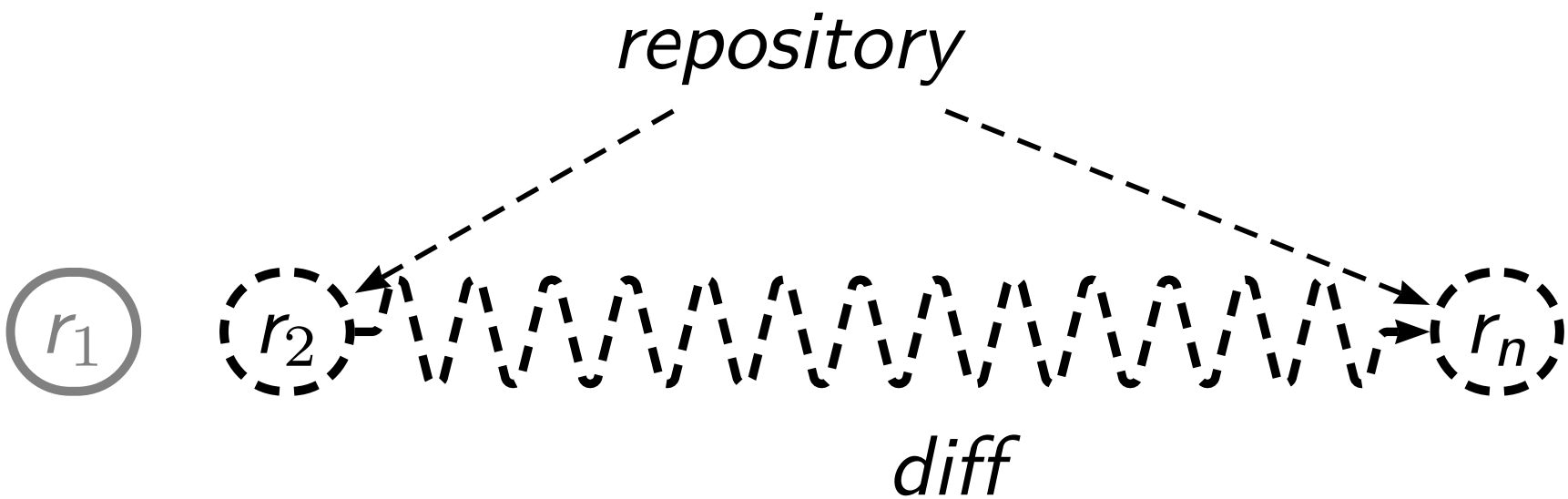
Concurrent changes, conflicts, solve, merge

I wrote a nice fix, I'm ready to commit.
Just before committing I **update** my working copy and notice a colleague solved the bug and committed.
My fix is a bit better — I have to solve (e.g. by **merging**) this **conflict**; then commit mine.



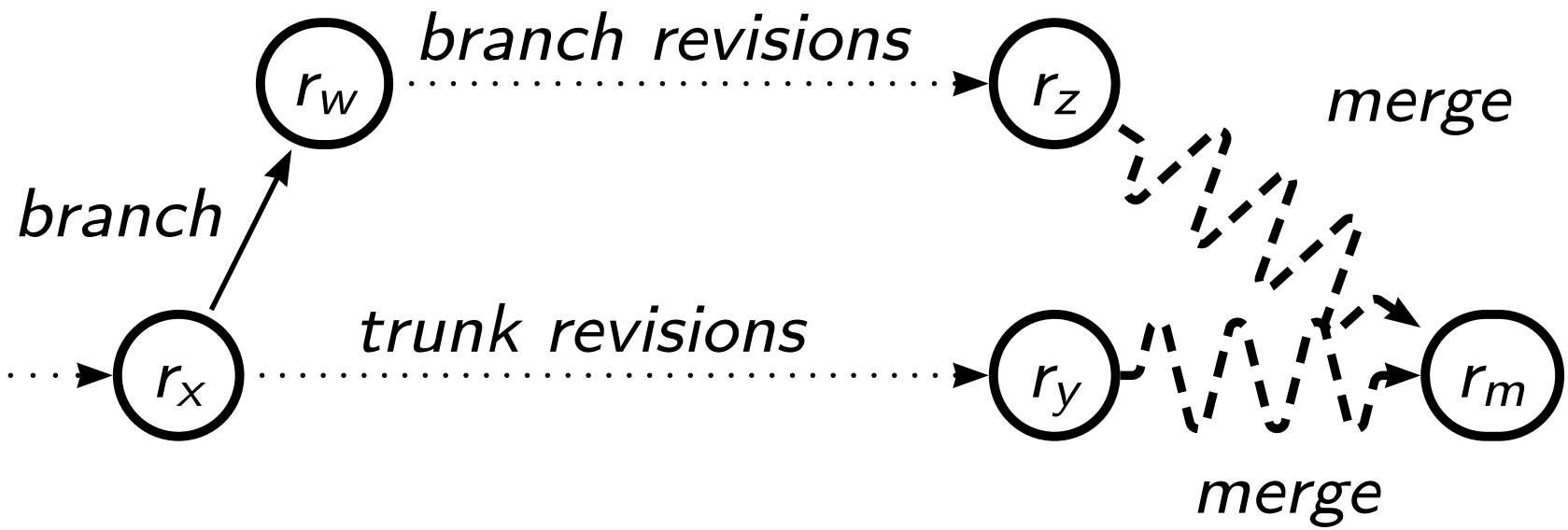
Diff'ing

How do our versions diverge ?
And wrt buggy version ?
Let me check the **diff**.



Branching and merging

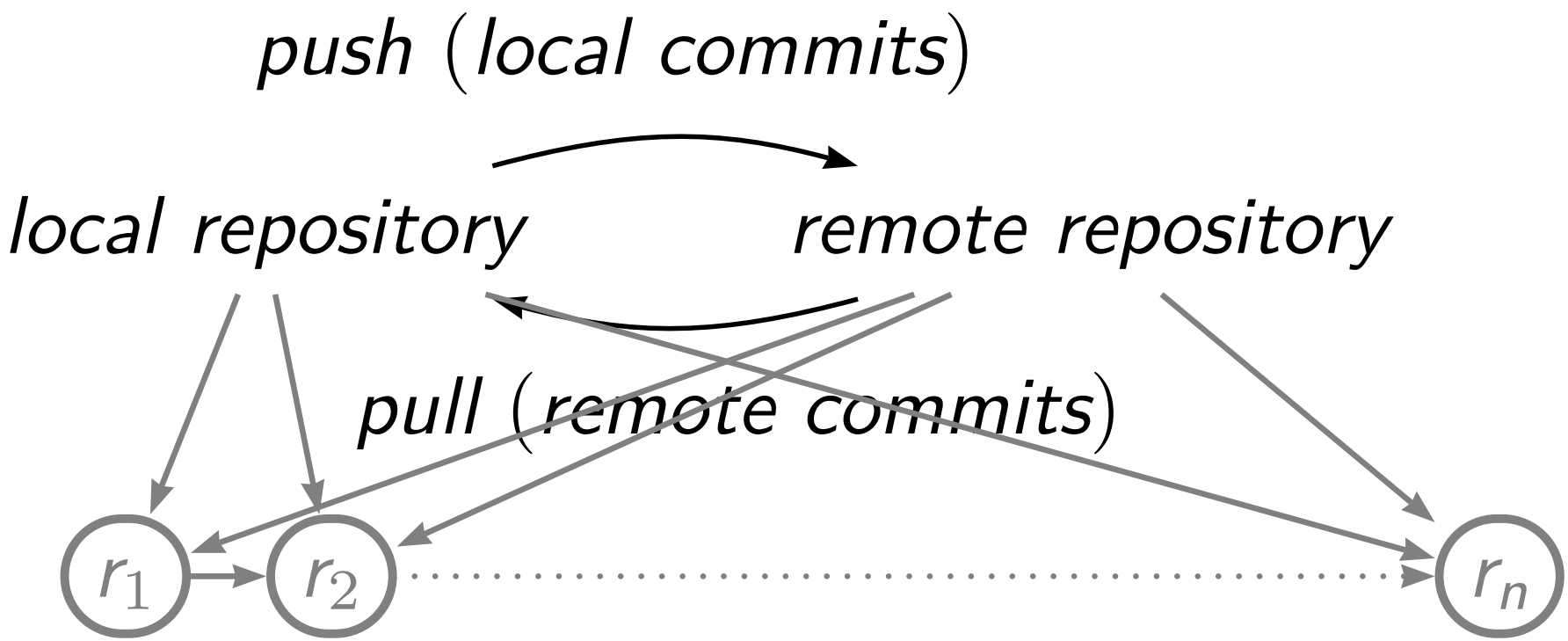
A **branch** begins as a copy of the tracked contents.
It evolves separately.
We may **merge** it.
Or we may discard it.



What if using a distributed version control system ?

Need to:

pull latest changes from remote repository to local one.
push my changes if no conflict detected.
Synchronization of revisions histories.



Do and don't do

Recommended:

- tracking changes of small text files
- cycle: edit ⇒ diff ⇒ commit ⇒ edit ...

Not recommended:

- mixing large and small files (e.g. sources and logs)
  - tracking changes is harder
  - better use separate repositories
- overly long/verbose log messages
- in writing software, too large *changesets*

Glossary, for our purposes

- **version control**: Tracking changes of a collection of files over time.
- **revision control**: aka **version control**.
- **repository**: Location containing the complete source code history of the project.
- **revision**: Current state ("snapshot") of a file or group of files in the repository.
- **checkout**: Request a working copy of a project from the repository.
- **commit**: (check in) Introduce changes on files in a sandbox back into the repository (repository is changed).
- **update**: Synchronize a developer's working copy to reflect the repository.
- **log message**: Developer's comment of a commit.
- **conflict**: Two developers make changes at the same point in the same file. Must be resolved manually.
- **push/pull**: Transmit commits to/from another repository.

Local usage

When ?

- learning
- testing
- beginning a project
- private stuff

Limitations ?

Not really: you can go *collaborative* afterwards, using e.g.:

- `svnadmin dump` / `svn import` / ...
- `git push` / `git clone` / ...

Local repository

Create a local repository

```
1 rm -fR                    git_lc
2 git init                  git_lc
```

```
1 Initialized empty Git repository in /local/user/
    git_lc/.git/
```

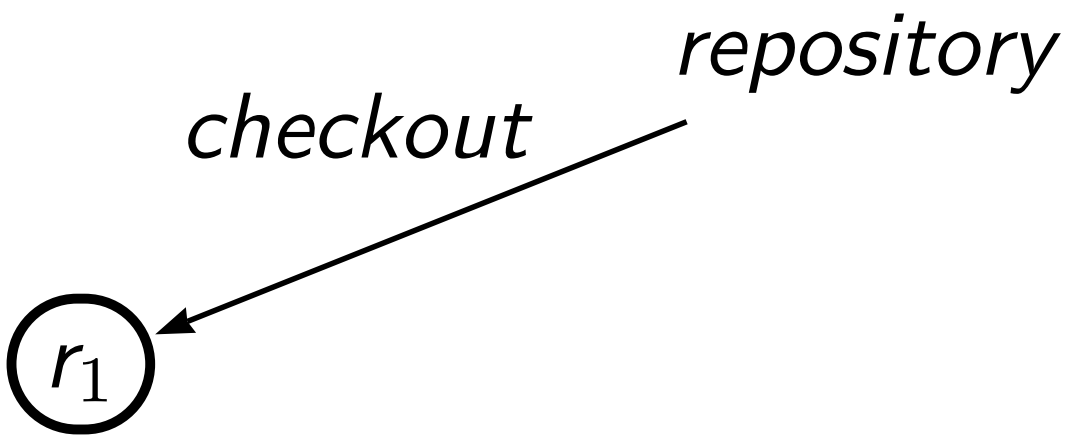
```
1 rm -fR                svn_repo
2 svnadmin create svn_repo
3 # No stdout output follows.
```

SVN: Check out a local copy

A *working copy*

```
rm -fR                              svn_lc
svn co file://`pwd`/svn_repo svn_lc
cd                                  svn_lc
svn status # no output
#svn info
```

```
Checked out revision 0.
```

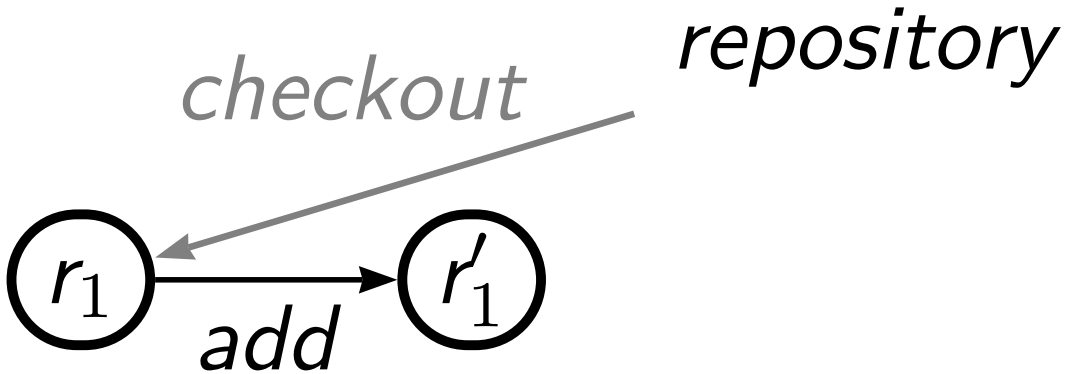


SVN: add

Add for addition

```
cd                                  svn_lc
touch          empty.txt
svn add        empty.txt
#svn status
#svn diff
```

```
A           empty.txt
```

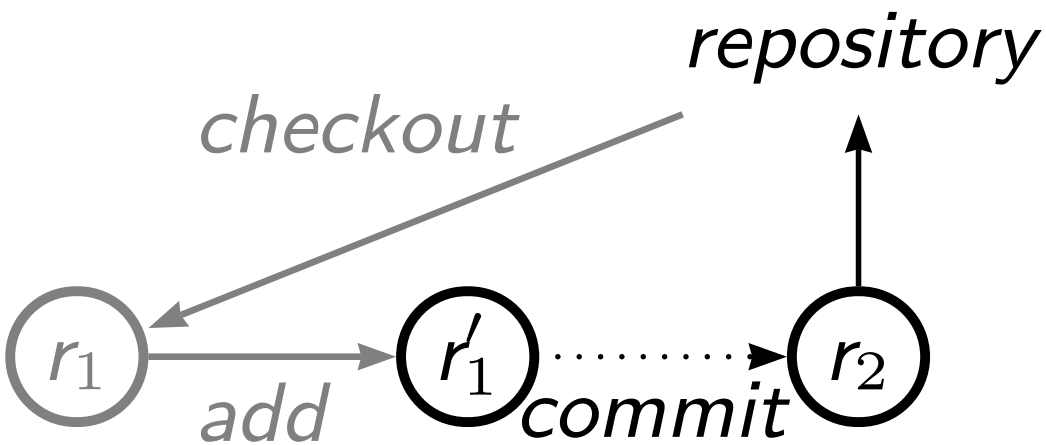


SVN: commit

Send revision to repository.

```
cd                                  svn_lc
svn commit  empty.txt -m 'added empty file'
svn status
```

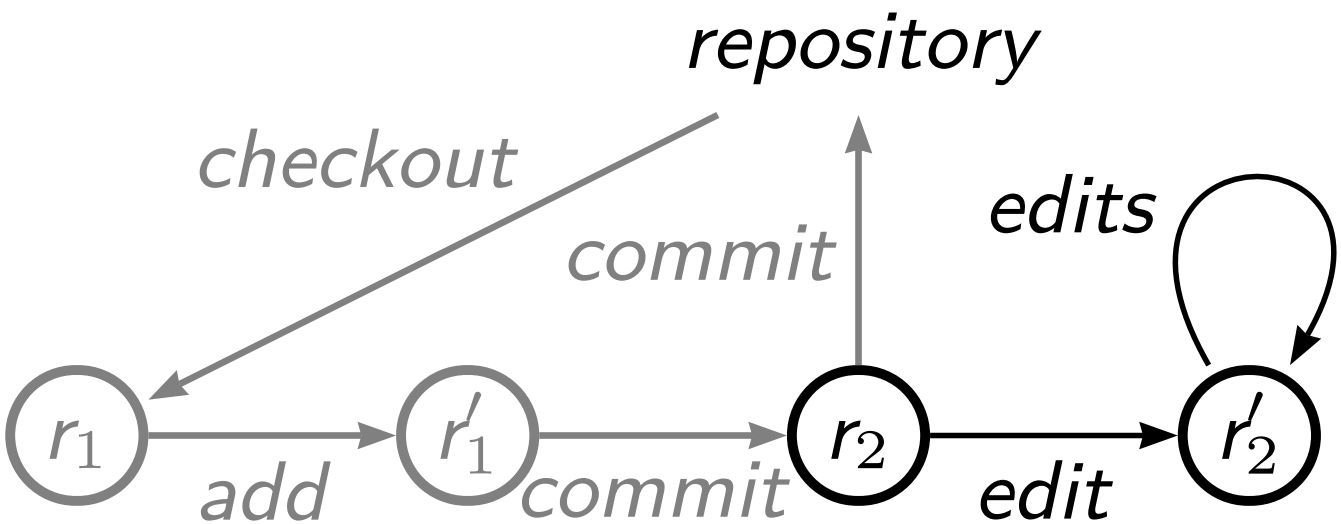
```
Adding          empty.txt
Transmitting file data .done
Committing transaction...
Committed revision 1.
```



SVN: edit

```
1 cd                                svn_lc
2 echo 'content' > empty.txt
3 svn status        empty.txt
```

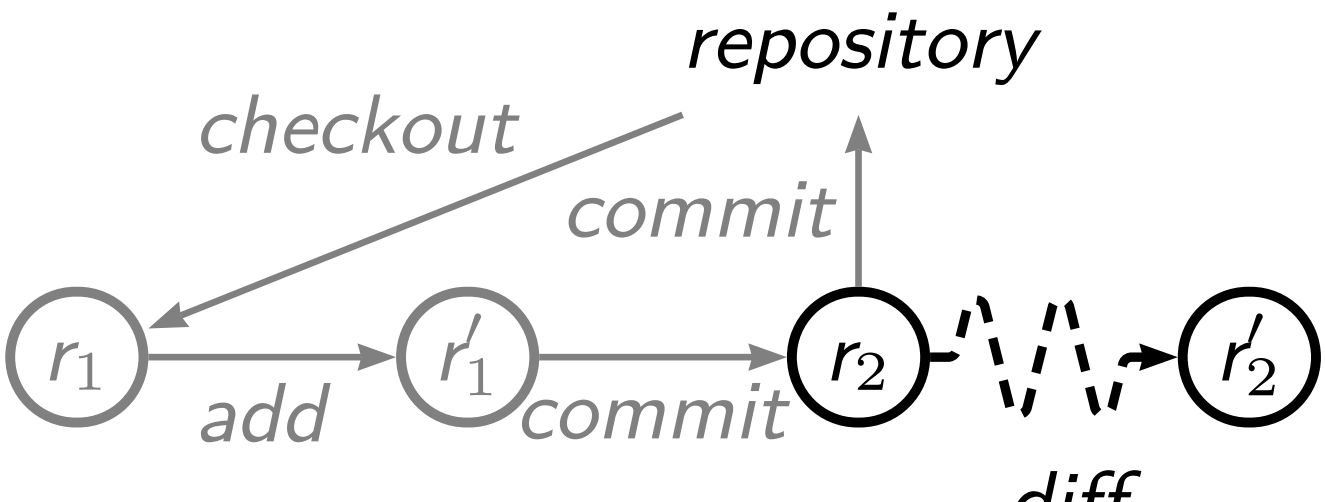
```
1 M       empty.txt
```



SVN: working copy diff

```
cd                                  svn_lc
svn diff          empty.txt
```

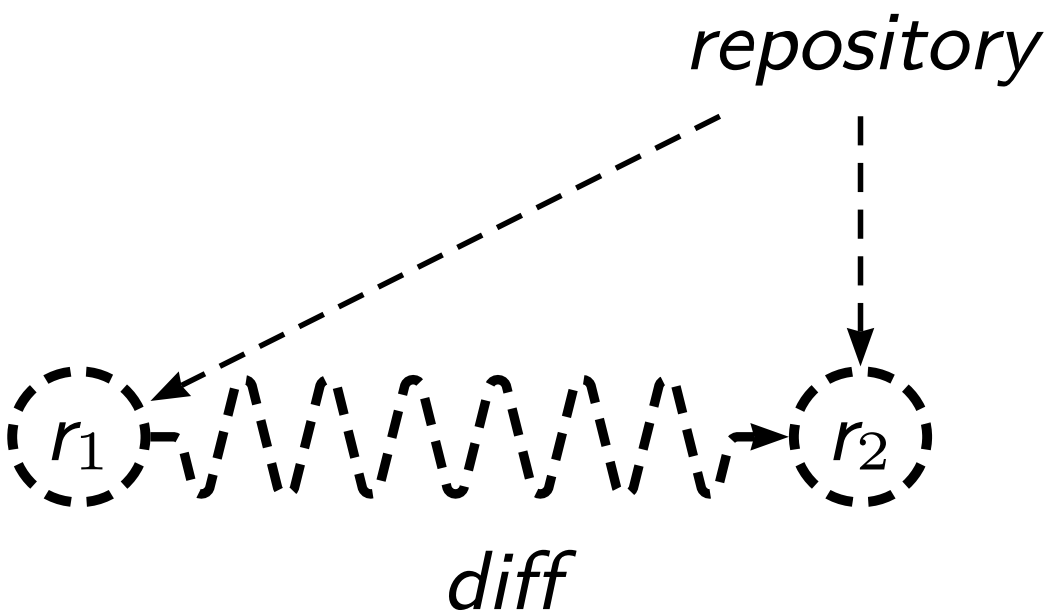
```
Index: empty.txt
===================================================================


--- empty.txt    (revision 1)
+++ empty.txt    (working copy)
@@ -0,0 +1 @@
+content
```



SVN: repo diff

```
1 cd                                    svn_lc
2 svn diff  -r0:1                       empty.txt
```

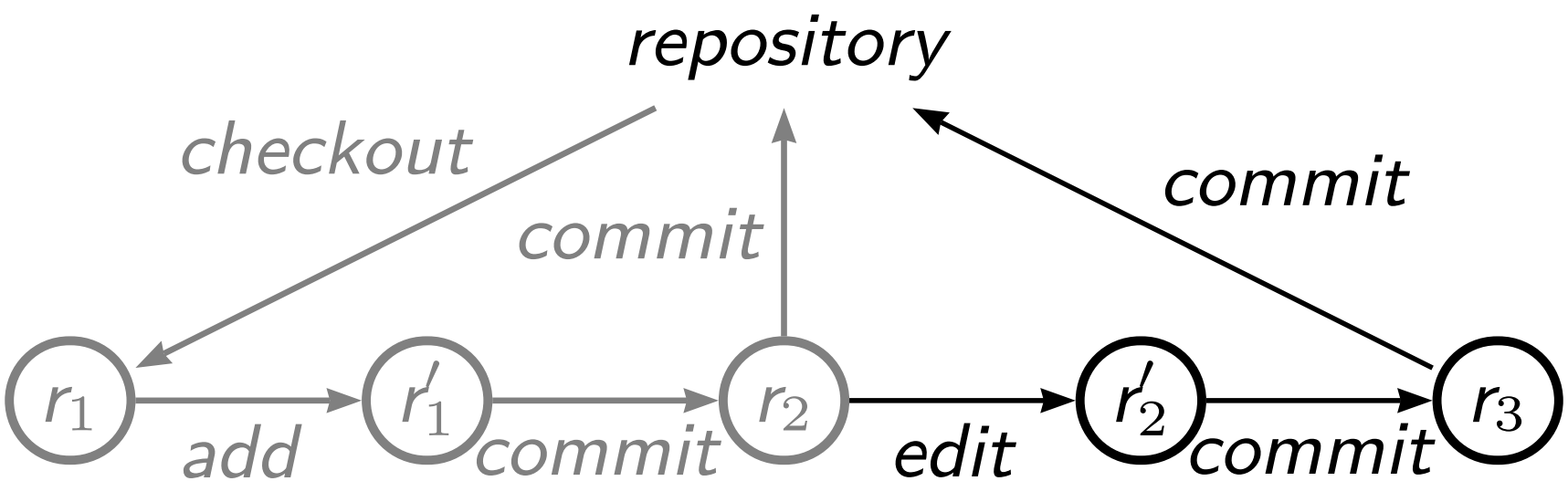
```
1 Index: empty.txt
2 ===================================================================
```



SVN: local work cycle

```
cd                                  svn_lc
svn commit  empty.txt -m 'added contents to file'
```

```
Sending        empty.txt
Transmitting file data .done
Committing transaction...
Committed revision 2.
```



Conventional tree organization

Trunk, Branches, Tags

```
svn ls https://svn.apache.org/repos/asf/subversion
```

```
...
trunk/      # everyday development tree
branches/   # several 'non-current' but maintained trees
tags/       # trees updated at a specific time
...
```

Conventional tree organization

Branches

```
svn ls https://svn.apache.org/repos/asf/subversion/branches
```

```
...
1.4.x/                    # versions
1.5.x/                    # ...
1.6.x/
1.6.x-CVE-2017-9800/ # developed to fix a bug
...
```

Typical SVN tree organization

Tags

```
svn ls https://svn.apache.org/repos/asf/subversion/tags
```

```
...
1.9.0/              # copies at specific points in time
1.9.0-alpha1/
1.9.0-alpha2/
1.9.0-beta1/
1.9.0-rc1/
1.9.0-rc2/
1.9.0-rc3/
...
```

What about GIT ?

Repository and working copy are both here.

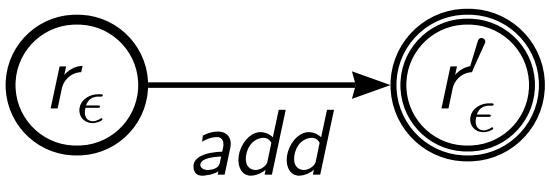
```
cd                                  git_lc
ls -ld -gG                          .git # repository here
git status # no output
#git log --oneline --decorate
#git info
```

```
drwxr-xr-x 7 119 Feb 28 21:29 .git
On branch master

Initial commit

nothing to commit (create/copy files and use "git add"
    to track)
```

SVN: repository is not in `.svn` (see `svn info`).

GIT: add

```
1 cd                                  git_lc
2 touch          empty.txt
3 git add        empty.txt          # put file in the index
```



The **index**

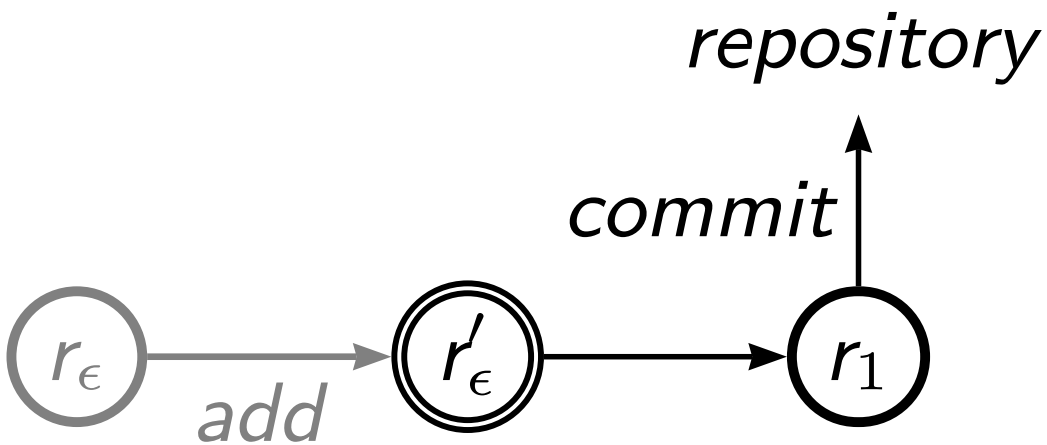
`git add` stages changes for next commit in the **index**.

`svn add` starts tracking files.

GIT: commit

```
cd                                  git_lc
git commit  empty.txt -m 'added empty file'
```

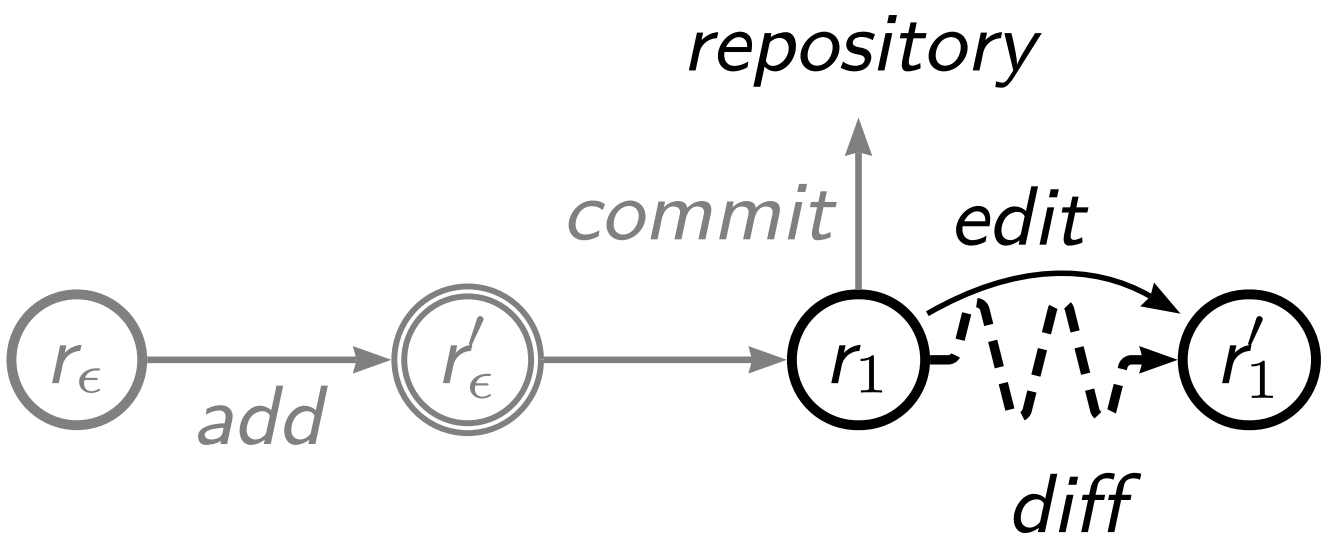
```
[master (root-commit) 78f77f9] added empty file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 empty.txt
```



GIT: edit and diff

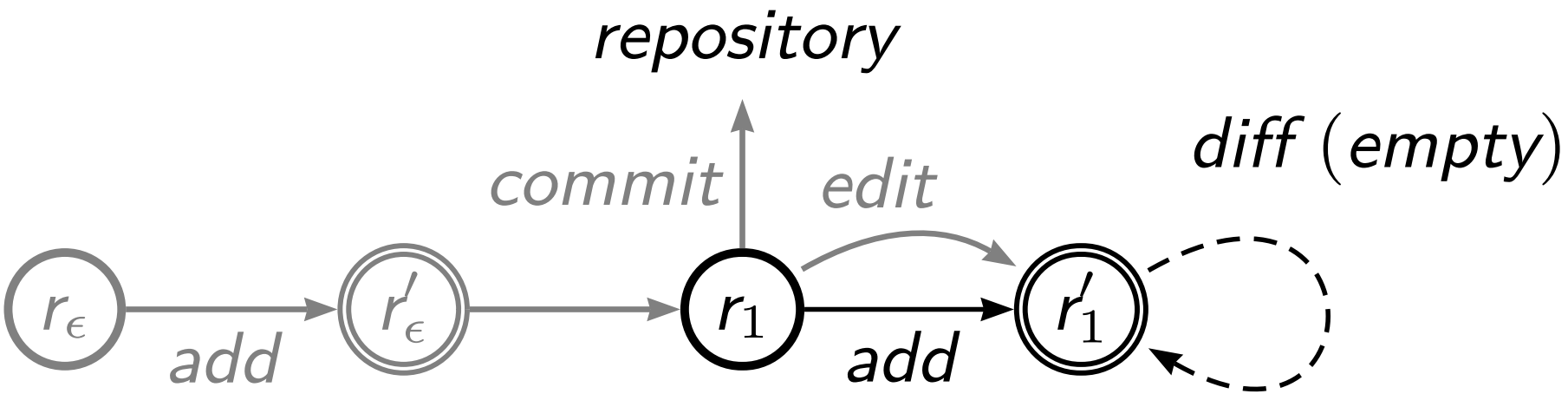
```
cd                  git_lc
echo 'content'  > empty.txt
git diff            # 'content'
```

```
diff --git a/empty.txt b/empty.txt
index e69de29..d95f3ad 100644
--- a/empty.txt
+++ b/empty.txt
@@ -0,0 +1 @@
+content
```



GIT: add and diff

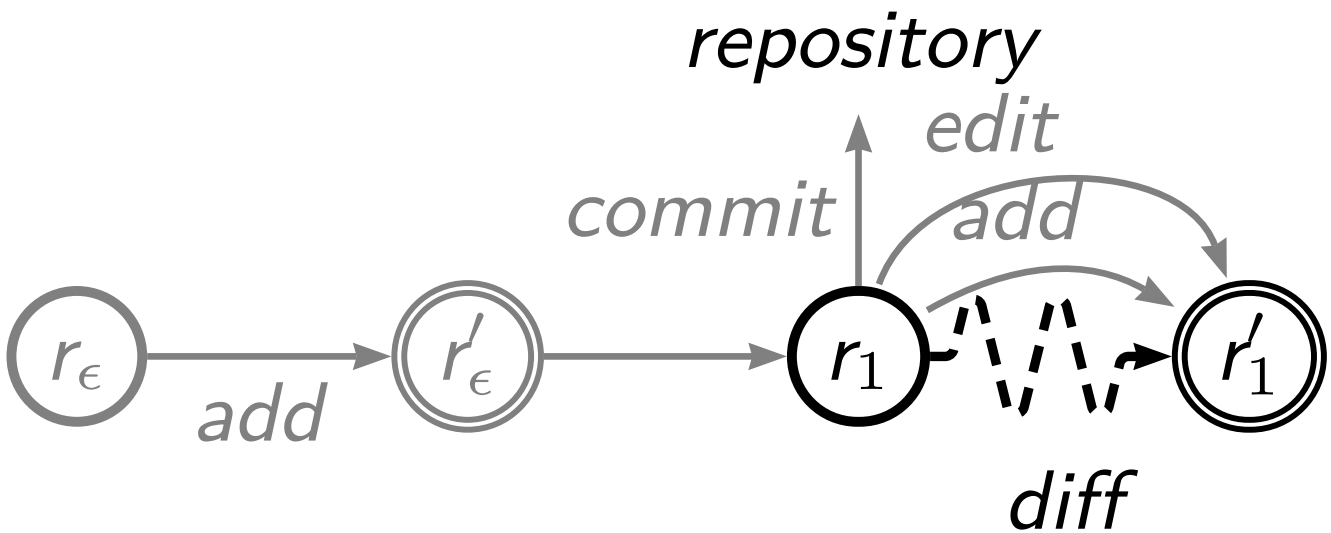
```
1 cd                    git_lc
2 git add               empty.txt
3 git diff              # empty: file contents are in the
      index
```



GIT: cached diff

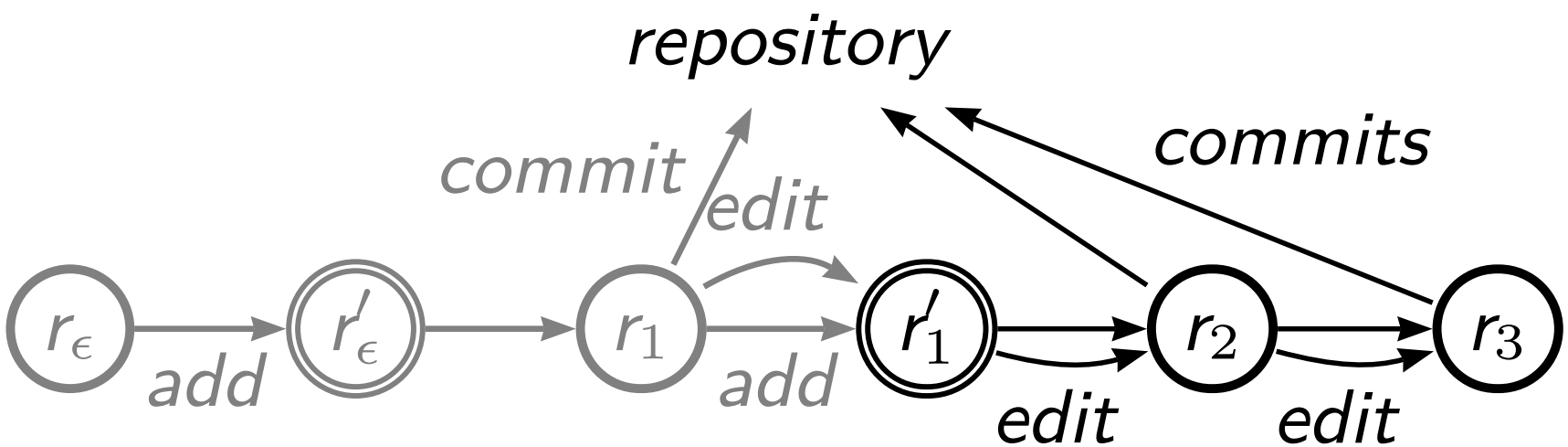
```
cd                    git_lc
git diff --cached empty.txt
```

```
diff --git a/empty.txt b/empty.txt
index e69de29..d95f3ad 100644
--- a/empty.txt
+++ b/empty.txt
@@ -0,0 +1 @@
+content
```



GIT: more commits

```
cd                    git_lc
echo 'more content'       >> empty.txt
git commit empty.txt -m 'added more content'       -q
echo 'even more content' >> empty.txt
git commit empty.txt -m 'added even more content' -q
```



GIT: what about *actions* history ?

```
cd                                  git_lc
git reflog
```

```
9c95374 HEAD@{0}: commit: added even more content
82702b0 HEAD@{1}: commit: added more content
78f77f9 HEAD@{2}: commit (initial): added empty file
```

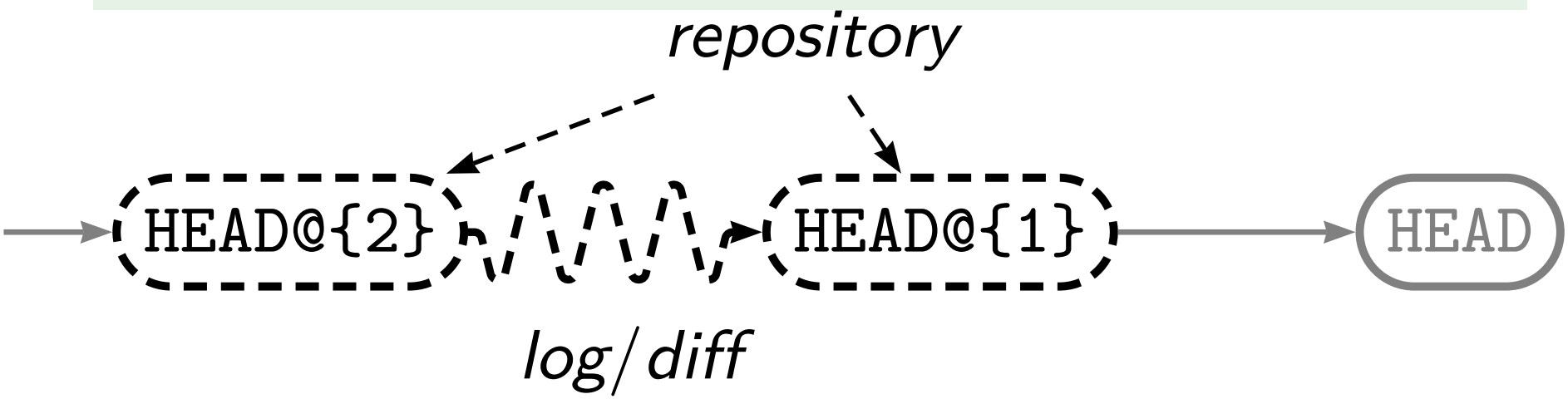
GIT: rev-parse'ing and HEAD

Well

There are **many** ways to refer to commits

```
1 cd                                  git_lc
2 git log 'HEAD@{2}..HEAD@{1}' # log message from second ancestor
      to last one
3 # git help rev-parse # see SPECIFYING REVISIONS section
```

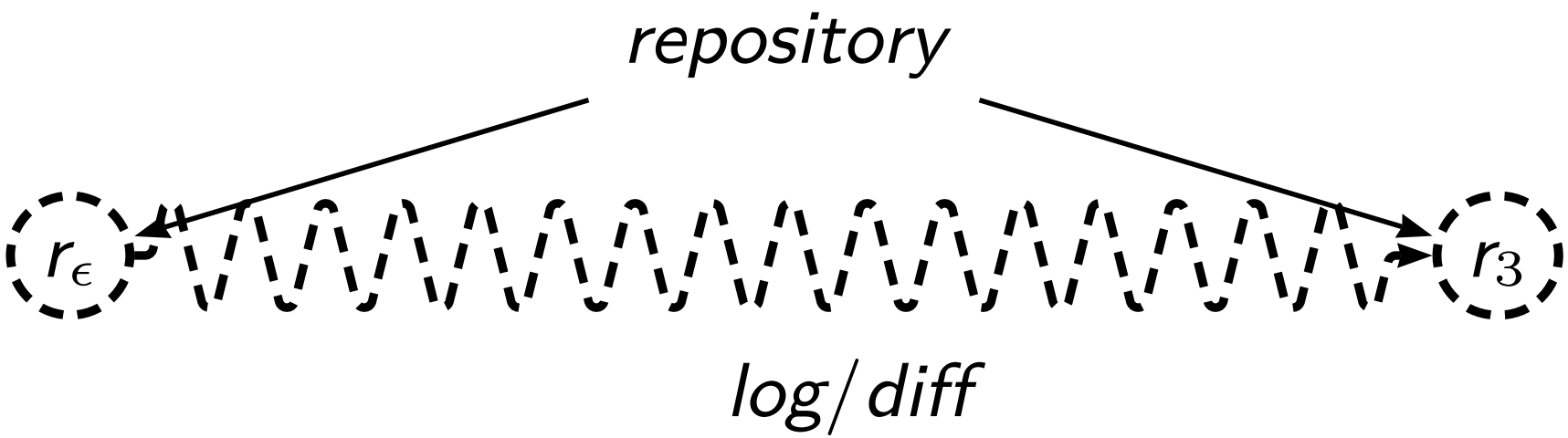
```
1 commit 82702b09b6d705405abe0d2d6bcf9ed02c2df680
2 Author: John Doe <john@doe.tld>
3 Date:   Thu Feb 28 21:30:17 2019 +0100
4
5     added more content
```



GIT: short log lines

```
cd                                git_lc
git log --oneline --decorate
```

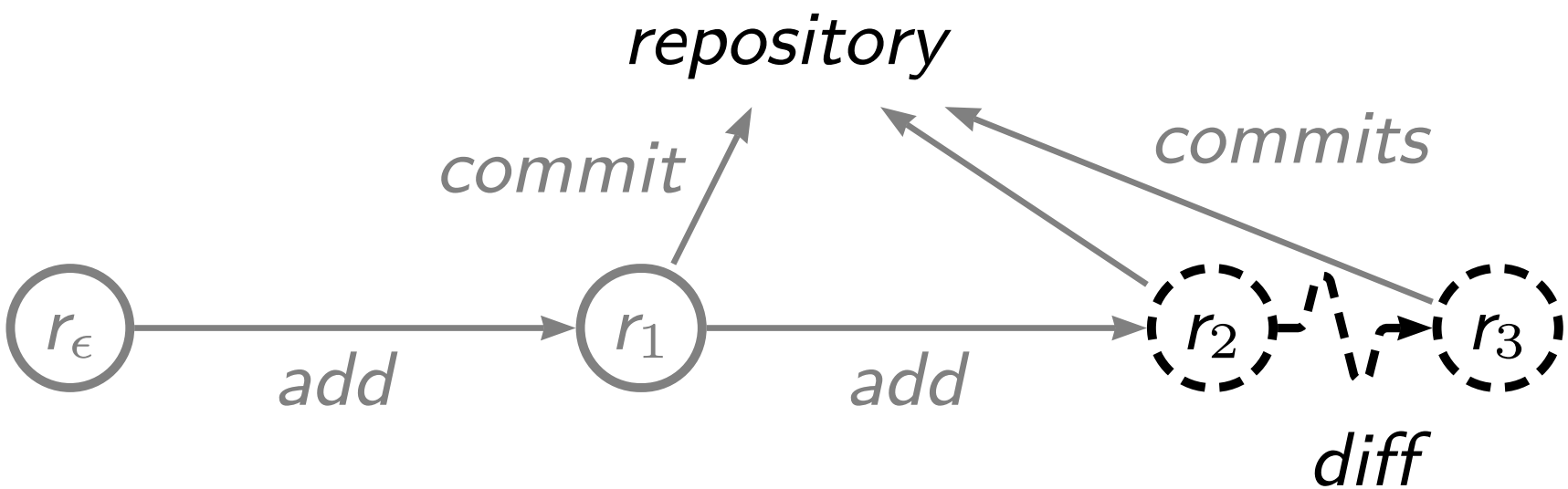
```
9c95374 (HEAD -> master) added even more content
82702b0 added more content
78f77f9 added empty file
```



GIT: last commit log

```
cd                                  git_lc
git log master~..master
```

```
commit 9c95374f0d66dc8b5ed434d28ad84ba5dcdf9f6d
Author: John Doe <john@doe.tld>
Date:   Thu Feb 28 21:30:17 2019 +0100

    added even more content
```

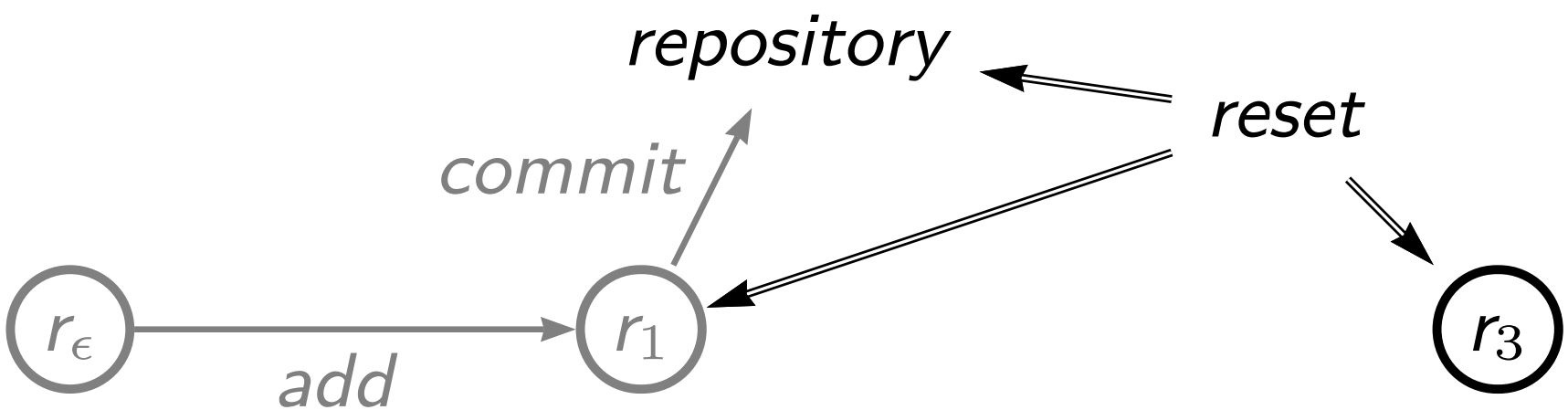


GIT: reset HEAD

A powerful and dangerous tool.

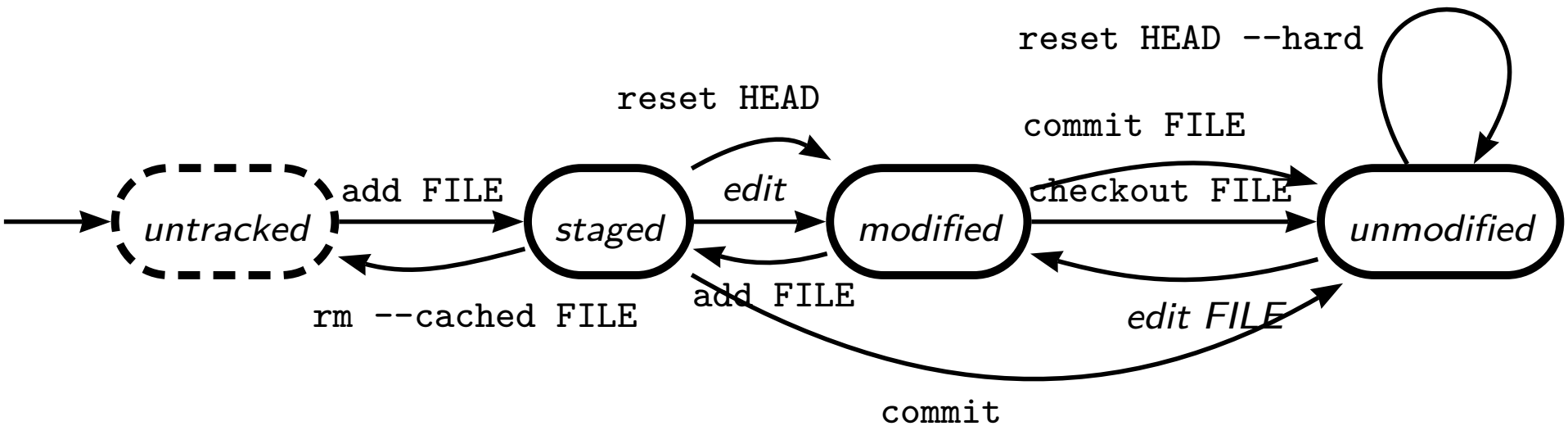
```
cd                                  git_lc
git log --oneline --decorate
git reset       HEAD~2  # "undo" two commits
```

```
9c95374 (HEAD -> master) added even more content
82702b0 added more content
78f77f9 added empty file
Unstaged changes after reset:
M       empty.txt
```



GIT: states of a file

Given a FILE in the working directory and executing `git ...` or *action*...



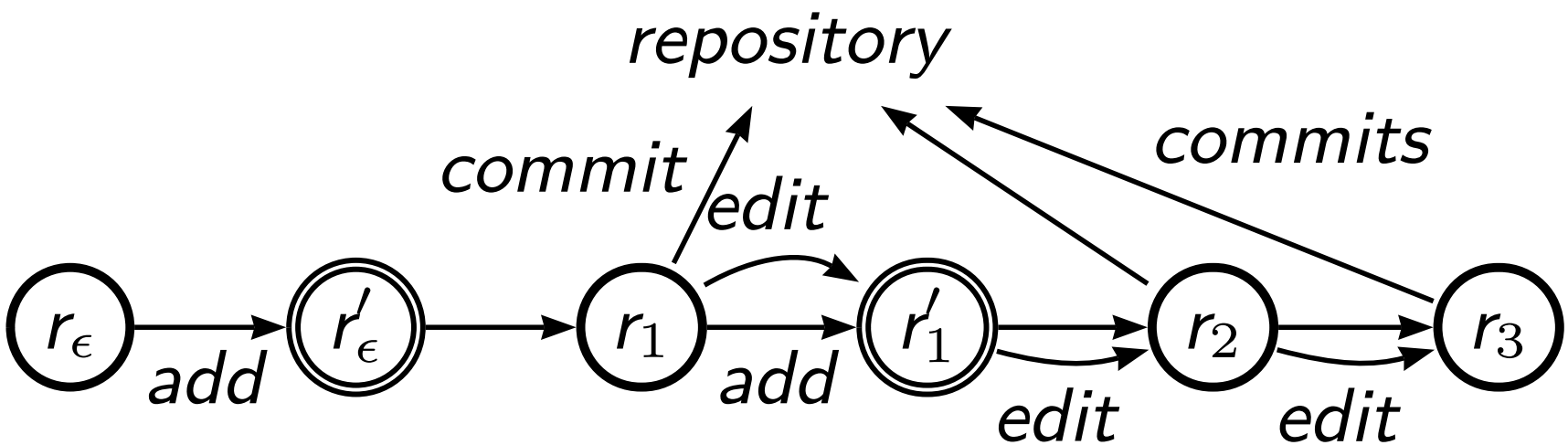
There are more transitions than these.

Local GIT usage recap

Basic use not so different from `svn`

Main source of confusion: GIT's index

`svn add` vs `git add`



Break!

Local copy of remote repository

Clone and checkout examples

```
rm -fR git_sc
git clone    https://github.com/schacon/example git_sc
```

```
Cloning into 'git_sc'...
```

```
rm -fR svn_lc
svn checkout https://github.com/schacon/example svn_lc
```

```
A     svn_lc/branches
A     svn_lc/trunk
A     svn_lc/trunk/README.md
A     svn_lc/trunk/classycat.png
Checked out revision 1.
```

however: `svn checkout` differs from `git checkout`

Routine work

Mostly the same:

similar syntax for add, commit, etc.

For big files, better remote operations!

`svn` and e.g. its `svn mv REMOTE-SRC REMOTE-DST` can be very useful

Pitfalls...

`git commit` is local!

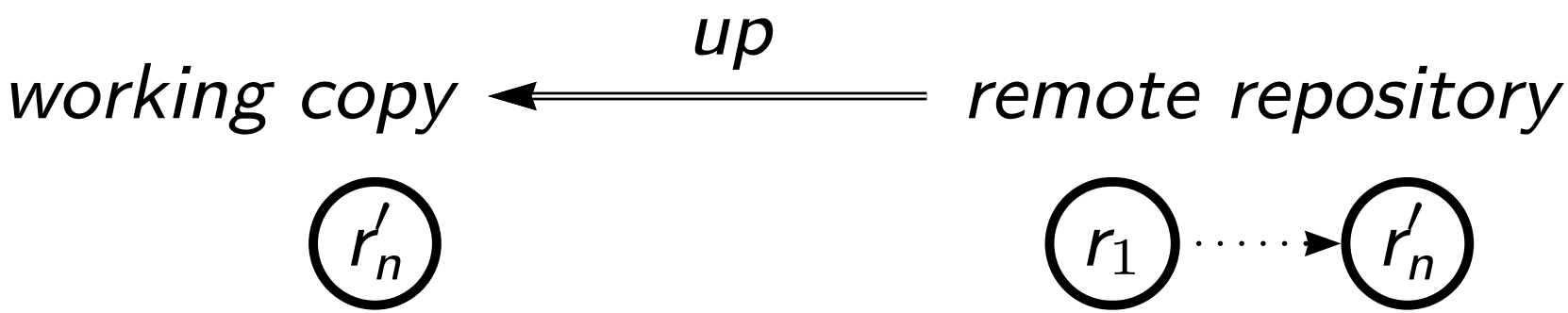
`svn commit` is remote!

SVN up

Remote repository is ahead

Synchronize local working copy

`svn up` fetches changes



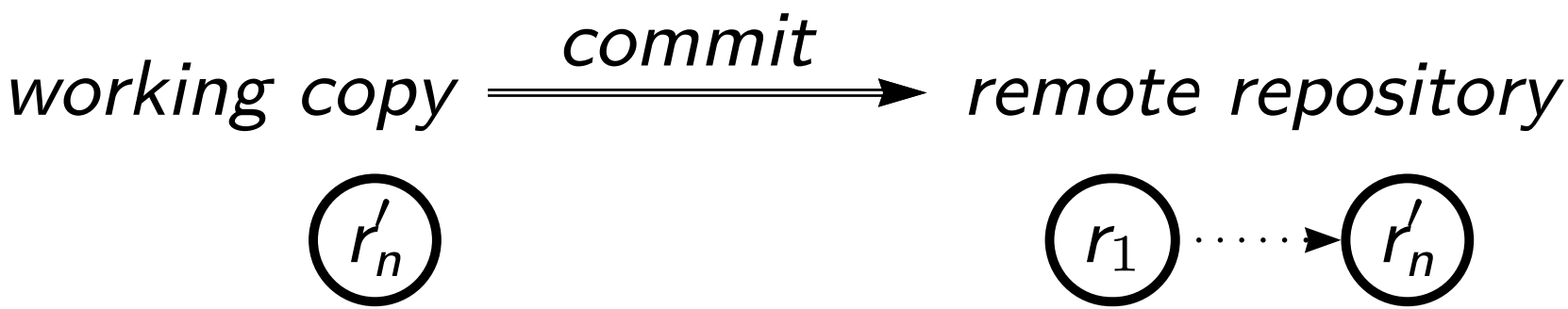
SVN commit

We have modified working copy

And we want to commit to a new revision.

Synchronize remote repository

If `svn commit` detects no conflict.

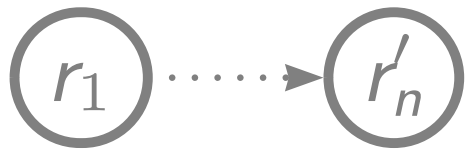


GIT: Local side initiates clone

There is no local repository yet

local repository

remote repository



Create synchronized local repository (and log messages, etc)

clone

local repository ⟸ *remote repository*

GIT: clone

Create two working copies and repositories

```
rm -fR                    git_wc git_ac
git clone  repo.git git_wc # working copy
git clone  repo.git git_ac # another working copy
```

```
Cloning into 'git_wc'...
done.
Cloning into 'git_ac'...
done.
```

GIT: set committer

For a clean log

```
cd                                  git_wc
git config --local --replace-all user.name 'John Doe'
git config --local --replace-all user.email 'john@doe.tld'
echo 'J. Doe' >  jd.txt
git add jd.txt    ; git commit -q -m 'my name'  .
git config --local --replace-all user.name 'John DOE'
echo 'J. DOE' >> jd.txt ; git commit -q -m 'now uppercase' .
git log master~2..master
```

```
commit 3b73de8d8851d3d7c7fd249e892cccd65cb1637d
Author: John DOE <john@doe.tld>
Date:   Thu Feb 28 21:30:52 2019 +0100

    now uppercase

commit 598eee79fa55156b89114fb0b442aa2f3091bb22
Author: John Doe <john@doe.tld>
Date:   Thu Feb 28 21:30:52 2019 +0100

    my name
```

GIT (and SVN): blame

Who modified when which line of a file ?

```
cd                                  git_wc
nl                                jd.txt
git blame master~2..master jd.txt
```

```
     1  J. Doe
     2  J. DOE
598eee79 (John Doe 2019-02-28 21:30:52 +0100 1) J. Doe
3b73de8d (John DOE 2019-02-28 21:30:52 +0100 2) J. DOE
```

GIT: remote

Tracked repositories

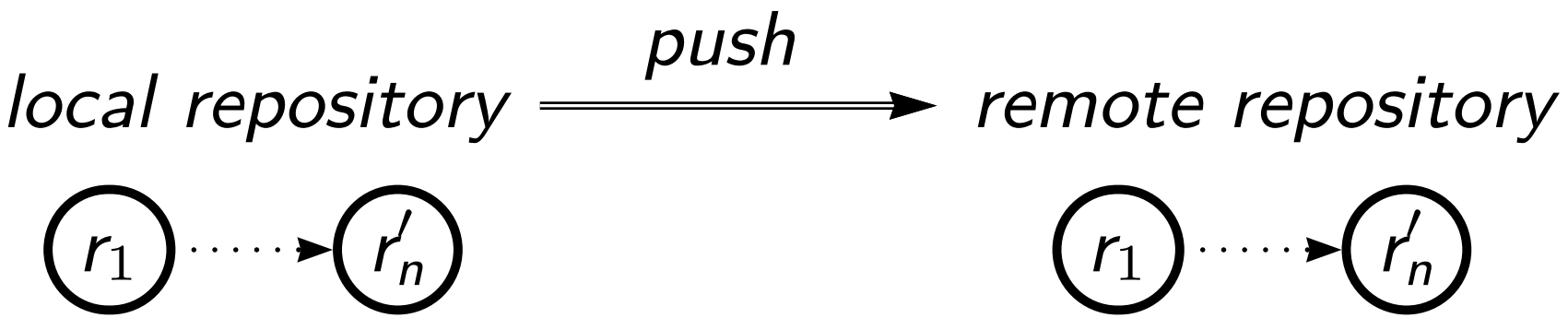
```
cd                                  git_wc
git remote -v
git log --graph --decorate --oneline
```

```
origin  /user/repo.git (fetch)
origin  /user/repo.git (push)
* 3b73de8 (HEAD -> master) now uppercase
* 598eee7 my name
* 95ce054 (origin/master, origin/HEAD) old stuff
* 78f77f9 added empty file
```

GIT: Local side initiates push

Remote repository is behind

Synchronize remote repository (and log messages, etc)



GIT: push

Sync remote: HEAD is not updated

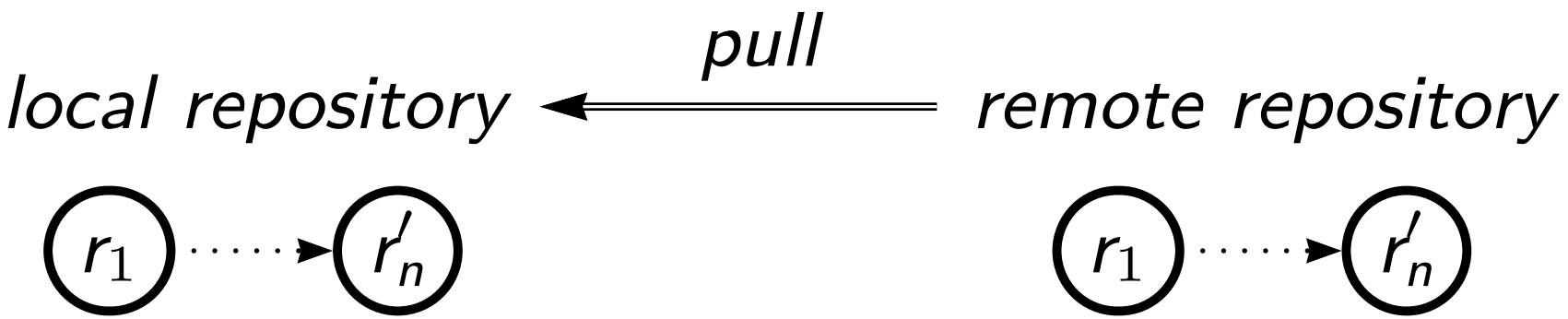
```
cd                                  git_wc
git log --graph --decorate --oneline
git push ../repo.git
git log --graph --decorate --oneline
```

```
* 3b73de8 (HEAD -> master) now uppercase
* 598eee7 my name
* 95ce054 (origin/master, origin/HEAD) old stuff
* 78f77f9 added empty file
To ../repo.git
   95ce054..3b73de8  master -> master
* 3b73de8 (HEAD -> master) now uppercase
* 598eee7 my name
* 95ce054 (origin/master, origin/HEAD) old stuff
* 78f77f9 added empty file
```

GIT: Local side initiates pull

Remote repository is ahead

Synchronize local repository (and log messages, etc)



GIT: pull

Sync from remote: HEAD is updated

```
cd  git_ac # remember ? outdated clone of repo.git
git log --graph --decorate --oneline master^1..master # HEAD
git pull
git log --graph --decorate --oneline
```

```
* 95ce054 (HEAD -> master, origin/master, origin/HEAD) old stuff
From /user/repo
   95ce054..3b73de8  master     -> origin/master
Updating 95ce054..3b73de8
Fast-forward
 jd.txt | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 jd.txt
* 3b73de8 (HEAD -> master, origin/master, origin/HEAD) now
    uppercase
* 598eee7 my name
* 95ce054 old stuff
* 78f77f9 added empty file
```

Break!

Hands-on sessions

Examples from page 64 onwards are meant for copy-paste.

Those on pages 80 to 94 are more advanced (take home).

Notice

Meant to work in sequence.

Tested on e.g. *svn 1.9.7* and *git 2.13.7*

SVN: Exercises

- Copy and paste freely from the following slides.
- Experiment
- Consult `svn help <subcommand>`

SVN: set up repository

```
rm -fR                  svn_repo
svnadmin create         svn_repo
svn co file://`pwd`/svn_repo
cd                      svn_repo
```

```
Checked out revision 0.
```

Try:

`svn ls` and its options

SVN: add ASCII file

```
cd                svn_repo
echo 'aeiou' > ascii.txt
svn add           ascii.txt
svn diff          ascii.txt # inspect difference
#date           > date.txt
#svn add          date.txt
```

```
A         ascii.txt
Index: ascii.txt
===================================================================


--- ascii.txt    (nonexistent)
+++ ascii.txt    (working copy)
@@ -0,0 +1 @@
+aeiou
```

Try:

Add a few files on your own, check `svn diff`.

SVN: commit ASCII file

```
cd                svn_repo
svn commit -m 'ascii file in.'
```

```
Adding            ascii.txt
Transmitting file data .done
Committing transaction...
Committed revision 1.
```

Try:

```
svn log
```

SVN: edit a file and commit

```
cd                  svn_repo
rev ascii.txt >> ascii.txt # reverses lines
svn diff
```

```
Index: ascii.txt
===================================================================


--- ascii.txt    (revision 1)
+++ ascii.txt    (working copy)
@@ -1 +1,2 @@
  aeiou
+uoiea
```

Try:

Edit file on your own and commit.

SVN: revert changes

```
cd                  svn_repo
echo 'O MAMMAMIA!'  > ascii.txt # Ochje
svn diff              ascii.txt
svn revert            ascii.txt
svn diff              ascii.txt # no output: fine
```

```
Index: ascii.txt
===================================================================


--- ascii.txt    (revision 1)
+++ ascii.txt    (working copy)
@@ -1 +1 @@
-aeiou
+O MAMMAMIA!
Reverted 'ascii.txt'
```

Try:

Edit and revert before committing.

SVN: add binary file

```
cd              svn_repo
echo 'äëïöü' > binary.txt # UTF-8 encoding
svn add         binary.txt
svn diff        binary.txt
```

```
A  (bin)  binary.txt
Index: binary.txt
===================================================================
Cannot display: file marked as a binary type.
svn:mime-type = application/octet-stream
Index: binary.txt
===================================================================
--- binary.txt  (nonexistent)
+++ binary.txt  (working copy)

Property changes on: binary.txt
___________________________________________________________________
Added: svn:mime-type
## -0,0 +1 ##
+application/octet-stream
\ No newline at end of property
```

Try:

Discover how to make `svn diff` work again with `svn help ps`

See notes on page 78. Extra: see `man ascii`. Extra: see `man utf8`.

GIT: Exercises

- Copy and paste freely from the following slides.
- Experiment
- Consult `git help <subcommand>`

GIT: Repo init

```
rm -fR      git_repo
git init    git_repo
cd          git_repo
git       status
```

```
Initialized empty Git repository in /local/user/git_repo/.git/
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

Notice

Differently than in SVN, this directory hosts a repository, too.

GIT: Add file

```
1 cd          git_repo
2 date        > date.txt
3 git add       date.txt
4 git diff
```

Try

`git diff` : any difference with SVN ?

GIT: Log

```
1 cd          git_repo
2 # git         log # try this
```

Try

`git log`: any difference with SVN ?

GIT: Graph view log

```
cd       git_repo
# TODO: branch and a few commits ...
git commit date.txt -m 'date file in'
git log --graph --oneline -- | tail -n 16
```

```
[master (root-commit) ba907c1] date file in
 1 file changed, 1 insertion(+)
 create mode 100644 date.txt
* ba907c1 date file in
```

GIT: amend a commit

```
cd       git_repo
echo 'Introduction.'     > intro.txt
git add                    intro.txt
git commit -q intro.txt -m 'Began writing intro.'
echo '...'               >> intro.txt
git commit -q intro.txt -m 'Edited iintro.'
git log --graph --decorate --oneline HEAD...HEAD~2
echo
git commit -q intro.txt -m 'Wrote introduction.' --amend
git log --graph --decorate --oneline HEAD...HEAD~2
#git push # send history to remote repository
```

```
* e956394 (HEAD -> master) Edited iintro.
* aef0728 Began writing intro.

* 587e283 (HEAD -> master) Wrote introduction.
* aef0728 Began writing intro.
```

Try:

`git help`, `... --amend --author <>`, `... --amend --date <>`

Not all bytes are ASCII

ASCII vs "binary" data

ASCII[4] (American Standard Code for Information Interchange) is a 7-bit encoding of the Latin alphabet and punctuation symbols.

```
# 7-bit encoded character, ASCII:
echo -n 'o' | xxd -c 2 -b # display as binary

# 8-bit encoded character, UTF-8:
echo -n 'ö' | xxd -c 2 -b
```

```
0000000: 01101111                  o
0000000: 11000011 10110110  ..
```

First bit of ASCII byte is unset

00000000 00000001 ... 01111111: ASCII
10000000 10000001 ... 11111111: ISO 8859-1 or other

A non-ASCII file may seem *binary* (see page 71).

[4] Extra: see `man ascii`. Extra: see `man charsets`.

Not all text files are ASCII

file: named sequence of bytes residing on a **file system**.[5]

```
1 echo 'echo is a shell command'
2 echo 'TEXT' > file.txt # write to file.txt
3 ls -l -gG     file.txt #          file.txt is a file
4 file          file.txt #                a text file
5 cat           file.txt #  let's read it
6 xxd -c 5 -b   file.txt #  let's check its bytes
```

```
1 echo is a shell command
2 -rw-r--r-- 1 5 Feb 28 21:31 file.txt
3 file.txt: ASCII text
4 TEXT
5 0000000: 01010100 01000101 01011000 01010100 00001010  TEXT.
```

Revision control with non-ASCII might bring surprises.

Will see this in action at page 71.

[5] Extra: see `man filesystems`.

More advanced useful exercises

- We recommend to experiment !
- For references, see page 94.

GIT bundle: offline backups

```
set -e
rm -fR git_offline; git init -q git_offline; cd git_offline
touch file.txt; git add file.txt; git commit -q -m "file in";
for i in `seq 1 10` ; do echo $i >> file.txt; git commit -q file.txt -m $i..; done
git bundle create bundled.bundle HEAD
cd -
rm -fR git_bundled
git bundle verify git_offline/bundled.bundle
git clone -q  git_offline/bundled.bundle git_bundled 2>&1 > /dev/null
cd git_bundled
#git log --oneline --decorate
git remote -v
```

```
/user
git_offline/bundled.bundle is okay
The bundle contains this ref:
82573f4b1fcf16fcc8a7648576f8c79a5424e0d2 HEAD
The bundle records a complete history.
Note: checking out '82573f4b1fcf16fcc8a7648576f8c79a5424e0d2'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

origin  /user/git_offline/bundled.bundle (fetch)
origin  /user/git_offline/bundled.bundle (push)
```

You can use bundle for backups or offline transfers.

GIT: branch and merge

```
set -e
cd git_offline
#
git checkout -b new_branch
echo 'branch material' >> file.txt
git commit file.txt -m 'branch committed'
#
git checkout master
echo 'master material' >> file.txt
git commit file.txt -m 'master committed'
#
git merge new_branch || true                # a merge strategy
sed -i 's/====/EEEEEE/g' file.txt           # a merge strategy
sed -i 's/.*HEAD$/LLLLLL/g' file.txt        # a merge strategy
sed -i 's/.*new_branch$/RRRRRR/g' file.txt  # a merge strategy
git add  file.txt
git commit -m conflict\ solved
git log --graph --oneline --decorate --all HEAD | head -n 6
```

```
Switched to a new branch 'new_branch'
[new_branch dc0c039] branch committed
 1 file changed, 1 insertion(+)
Switched to branch 'master'
[master 9e48a67] master committed
 1 file changed, 1 insertion(+)
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.
[master df0001f] conflict solved
*   df0001f (HEAD -> master) conflict solved
|\
| * dc0c039 (new_branch) branch committed
* | 9e48a67 master committed
|/
* 82573f4 10..
```

SVN and GIT from same dir

```
rm -fR hybd         svn_rep
svnadmin create     svn_rep # svn_rep:local SVN repository
svn co file://`pwd`/svn_rep hybd
cd                      hybd # hybrid dir: local svn_rep copy...
git init                     # ...and GIT repo
date > date.txt
svn add    date.txt
svn commit date.txt -m 'svn commit of date file'
svn up
git add    date.txt
git commit date.txt -m 'git commit of date file' -q
svn log | perl -l40pe 's/^-+/\n/' # one line log
git log --graph --oneline -- | tail -n 16
```

```
Checked out revision 0.
Initialized empty Git repository in /local/user/hybd/.git/
A         date.txt
Adding         date.txt
Transmitting file data .done
Committing transaction...
Committed revision 1.
Updating '.':
At revision 1.

 r1 | di36pef | 2019-02-28 21:31:12 +0100 (Thu, 28 Feb 2019) | 1 line  svn commit of date file
 * af11799 git commit of date file
```

An SVN local copy with a GIT repository.

SVN and GIT from same dir

```
cd                          hybd # hybrid dir: local svn_rep copy...
date +%s > date.txt
svn diff
echo
git diff
```

```
Index: date.txt
===================================================================
--- date.txt    (revision 1)
+++ date.txt    (working copy)
@@ -1 +1 @@
-Thu Feb 28 21:31:12 CET 2019
+1551385873

diff --git a/date.txt b/date.txt
index ecd9a80..62462cf 100644
--- a/date.txt
+++ b/date.txt
@@ -1 +1 @@
-Thu Feb 28 21:31:12 CET 2019
+1551385873
```

SVN and GIT from same dir

```
cd hybd
ls -ld -gG .git
ls -ld -gG .svn
```

```
drwxr-xr-x 8 166 Feb 28 21:31 .git
drwxr-xr-x 4 96 Feb 28 21:31 .svn
```

This usage style is good for practice with SVN and GIT.

Careful

Need to keep coherence of `git` and `svn` metadata manually.

By the way, you might also be interested into `git svn`.

Why not Mercurial, too ?

```
cd hybd
hg init
hg add    date.txt
echo -e "[ui]\n""username=<user@local.tld>" >> .hg/hgrc # hg config -e -l
hg commit date.txt -m 'hg commit of date file' -q
hg log
```

```
changeset:   0:8570a7590a4f
tag:         tip
user:        <user@local.tld>
date:        Thu Feb 28 21:31:13 2019 +0100
summary:     hg commit of date file
```

This usage style enables practice with Mercurial, too.

Careful

Redundant contents

```
cd hybd;                              # our hybrid SVN local copy and GIT repo
for i in `seq 1 3`; do                # for i one to three
  bs=1K;                              # block size: 1 kibibytes
  #bs=100M;                          # block size: 100 mibibytes
  src=/dev/zero;                      # zero   bytes source, e.g. 0000...
  #src=/dev/urandom;                  # random bytes source, e.g. 0110...
  of=zeros.$bs.$i.bin;
  dd if=$src of=$of bs=$bs count=1 status=noxfer; # add 3 1K-sized files
  du -h $of
  git add    $of ; git commit -q $of -m Added\ $of; du -sh .git;
  svn add -q $of ; svn commit -q $of -m Added\ $of; du -sh ../svn_rep;
done
```

```
4.0K    zeros.1K.1.bin
104K    .git
136K    ../svn_rep
4.0K    zeros.1K.2.bin
112K    .git
144K    ../svn_rep
4.0K    zeros.1K.3.bin
120K    .git
152K    ../svn_rep
```

Try substituting 1K with 100M.[6] Any difference ?
And 3 vs 300 ?

What about `/dev/urandom` instead of `/dev/zero` ?

[6] Extra: see `man units`.

GIT bisect

```
cd hybd;                                # our hybrid SVN local copy and GIT repo
cat > test.sh << EOF
#!/bin/bash
if test -f zeros.1K.1.bin ; then exit 1 ; else exit 0; fi
EOF
chmod +x test.sh
git log --oneline;
git tag -f Orig ; git tag -f BadRef HEAD; git tag -f  GoodRef BadRef~~~;
git bisect reset ;
git bisect start  ;
git bisect bad  BadRef;
git bisect good GoodRef;
git bisect run ./test.sh;
git co Orig;
```

```
6d934c9 Added zeros.1K.3.bin
4e10015 Added zeros.1K.2.bin
ec31b07 Added zeros.1K.1.bin
af11799 git commit of date file
We are not bisecting.
Bisecting: 0 revisions left to test after this (roughly 1 step)
[4e100150e16495f810bd5aa0c83778caff11b63f] Added zeros.1K.2.bin
running ./test.sh
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[ec31b075a4db913276c90e0035bdf92fb6263647] Added zeros.1K.1.bin
running ./test.sh
ec31b075a4db913276c90e0035bdf92fb6263647 is the first bad commit
commit ec31b075a4db913276c90e0035bdf92fb6263647
Author: Michele Martone <michele.martone@lrz.de>
Date:   Thu Feb 28 21:31:13 2019 +0100

    Added zeros.1K.1.bin

:000000 100644 0000000000000000000000000000000000000000 06d7405020018ddf3cacee90fd4af10487da3d20 A
            zeros.1K.1.bin
bisect run success
M       date.txt
```

SVN: Tracking diff after rename

```
#set -e;
rm -fR svn_mv_repo svn_mv && svnadmin create svn_mv_repo;
svn checkout -q file://`pwd`/svn_mv_repo svn_mv && cd svn_mv
mkdir old-dir new-dir
svn add old-dir new-dir    -q
svn commit -m 'added dirs' -q
touch          old-dir/old.txt; svn add -q old-dir/old.txt; svn commit -q old-dir/old.txt -m  "add old
       ".
for i in `seq 1 10`; do
 echo 'old' >>  old-dir/old.txt;                      svn commit -q old-dir/old.txt -m "${i}th edit
        old".
 svn mv -q old-dir/old.txt new-dir/new.txt;
 svn commit -q -m 'old out, new in.' new-dir/new.txt old-dir/old.txt;
 #svn commit -q -m 'new in.' new-dir/new.txt
 #svn commit -q -m 'old out.'  old-dir/old.txt;
 echo 'new' >>  new-dir/new.txt;                      svn commit -q new-dir/new.txt -m "${i}th edit
        new".
 svn mv -q new-dir/new.txt old-dir/old.txt;
 svn commit -q -m 'new out, old in.' old-dir/old.txt new-dir/new.txt;
 #svn commit -q -m 'old in.'  old-dir/old.txt;
 #svn commit -q -m 'new out.' new-dir/new.txt
done
svn up
svn log         |grep edit|wc -l
svn log old-dir/old.txt |grep edit|wc -l # same count
```

```
Updating '.':
At revision 42.
20
20
```

Commit order matter.

GIT: Careful with renaming

```
#set -e;
rm -fR git_rename && git init -q git_rename && cd git_rename
mkdir old-dir new-dir
touch          old-dir/old.txt; git add old-dir/old.txt; git commit -q old-dir/old.txt -m  add\ old.
for i in `seq 1 10`; do
 echo 'old' >>  old-dir/old.txt;                      git commit -q old-dir/old.txt -m ${i}th\ edit\
       old.
 git mv old-dir/old.txt new-dir/new.txt;
 git commit -q -m 'old out, new in.' old-dir/old.txt new-dir/new.txt;
 #git commit -q -m 'old out' old-dir/old.txt;
 #git commit -q -m 'new in'  new-dir/new.txt;

 echo 'new' >>  new-dir/new.txt;                      git commit -q new-dir/new.txt -m ${i}th\ edit\
       new.
 git mv new-dir/new.txt old-dir/old.txt;
 git commit -q -m 'new out, old in.'  old-dir/old.txt new-dir/new.txt;
 #git commit -q -m 'new out' new-dir/new.txt;
 #git commit -q -m  'old in' old-dir/old.txt;
done
git log --graph --oneline                  |grep edit |wc -l
git log --graph --oneline          old-dir/old.txt |grep edit |wc -l # anything missing?
git log --graph --oneline --follow old-dir/old.txt |grep edit |wc -l # ok
git config --local --replace-all log.follow 1               # always follow
git log --graph --oneline          old-dir/old.txt |grep edit |wc -l # ok
```

```
20
10
20
20
```

SVN: Tracking diff after rename

```
#set -e;
rm -fR svn_mv_repo svn_mv && svnadmin create svn_mv_repo;
svn checkout -q file://`pwd`/svn_mv_repo svn_mv && cd svn_mv
touch          old.txt; svn add -q old.txt; svn commit -q old.txt -m  "add old".
for i in `seq 1 10`; do
 echo 'old' >>  old.txt;                      svn commit -q old.txt -m "${i}th edit old".
 svn mv -q old.txt new.txt;
 svn commit -q -m 'old out, new in.' new.txt old.txt;
 #svn commit -q -m 'new in.' new.txt
 #svn commit -q -m 'old out.'  old.txt;
 echo 'new' >>  new.txt;                      svn commit -q new.txt -m "${i}th edit new".
 svn mv -q new.txt old.txt;
 svn commit -q -m 'new out, old in.' old.txt new.txt;
 #svn commit -q -m 'old in.'  old.txt;
 #svn commit -q -m 'new out.' new.txt
done
svn up
svn log         |grep edit|wc -l
svn log old.txt |grep edit|wc -l # same count
```

```
Updating '.':
At revision 41.
20
20
```

Commit order matter.

GIT: Careful with renaming

```
#set -e;
rm -fR git_rename && git init -q git_rename && cd git_rename
touch          old.txt; git add old.txt; git commit -q old.txt -m  add\ old.
for i in `seq 1 10`; do
 echo 'old' >>  old.txt;                      git commit -q old.txt -m ${i}th\ edit\ old.
 git mv old.txt new.txt;
 git commit -q -m 'old out, new in.' old.txt new.txt;
 #git commit -q -m 'old out' old.txt;
 #git commit -q -m 'new in'  new.txt;

 echo 'new' >>  new.txt;                      git commit -q new.txt -m ${i}th\ edit\ new.
 git mv new.txt old.txt;
 git commit -q -m 'new out, old in.'  old.txt new.txt;
 #git commit -q -m 'new out' new.txt;
 #git commit -q -m  'old in' old.txt;
done
git log --graph --oneline                    |grep edit |wc -l
git log --graph --oneline          old.txt |grep edit |wc -l # anything missing?
git log --graph --oneline --follow old.txt |grep edit |wc -l # ok
git config --local --replace-all log.follow 1               # always follow
git log --graph --oneline          old.txt |grep edit |wc -l # ok
```

```
20
10
20
20
```

Caveat Emptor

Ok, but which is better ?

It depends

Requirement	svn	git	comment
documentation	+	-	transparency and clarity
UI design/tools clarity	+	-	coherence of naming
offline/distributed work	-	**+**	can do offline commits
flexibility	-	+	GIT's *porcelain* vs *plumbing* tools
migrating from	+	-	conceptual + practical (converters)
when repository is huge	**+**	-	remote central vs replicated
ease of use	+	-	intuitiveness
branching	-	+	*proper branching*
editable history	-	+	`rebase, amend`
feature mattering you	**?**	**?**	**learn basics, test, simulate, choose**

Check for yourself based on your requirements.
You might check e.g. **hg** (Mercurial), too. See page 83.

References

- `man git`, `man gittutorial`, `man svn`, /usr/share/doc/git/html
- `http://www-cs-students.stanford.edu/~blynn/gitmagic/`
- `https://github.com/progit/progit2/releases/download/2.1.44/progit.pdf`
- `http://svnbook.red-bean.com/en/1.7/svn-book.pdf`
- `https://www.kernel.org/pub/software/scm/git/docs/user-manual.html`
- `http://subversion.apache.org/`
- `svn co https://svn.apache.org/repos/asf/subversion/trunk subversion`
  (a hundred MB...)
- `git clone git://git.kernel.org/pub/scm/git/git.git git`
  (a hundred MB...)
- `git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git`
  (a few GB...)
- `http://gitlab.lrz.de/`