Working with SVN and git

Florido Paganelli
Lund University
florido.paganelli@hep.lu.se

Outline

- What are version/revision control systems
  - Generic concepts of version/revision systems
- SVN
  - Generic concepts of SVN
  - SVN tutorial

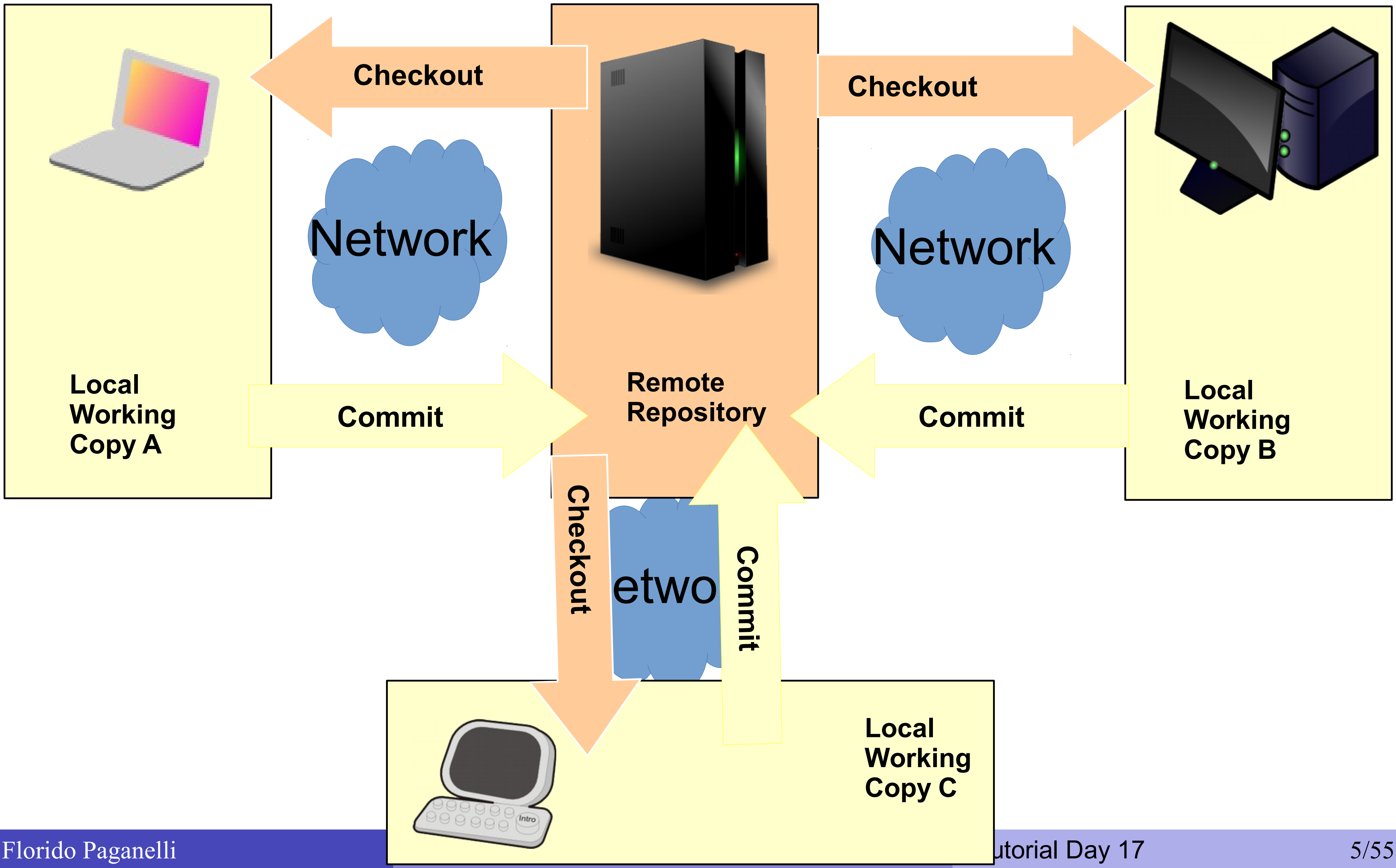
Why version/revision systems?

- Say you wrote some piece of code, and you give it to a colleague.
- Your colleague discovers it contains errors. You must fix them!
- You take the files and make a change trying to fix. Unfortunately it was a big mistake and you had to change quite lots of code.
- At the same time, your colleague really needed to work with these, so he made other changes himself.
- Once you compile, you discover that something else that worked before is not working anymore.
- What went wrong? Would be nice if you could compare what you changed...what your colleague changed... but, you overwrote all your old code!
- **Solution:** make a backup copy before every change!
- Version systems make it easy to backup , share and compare **changes**.

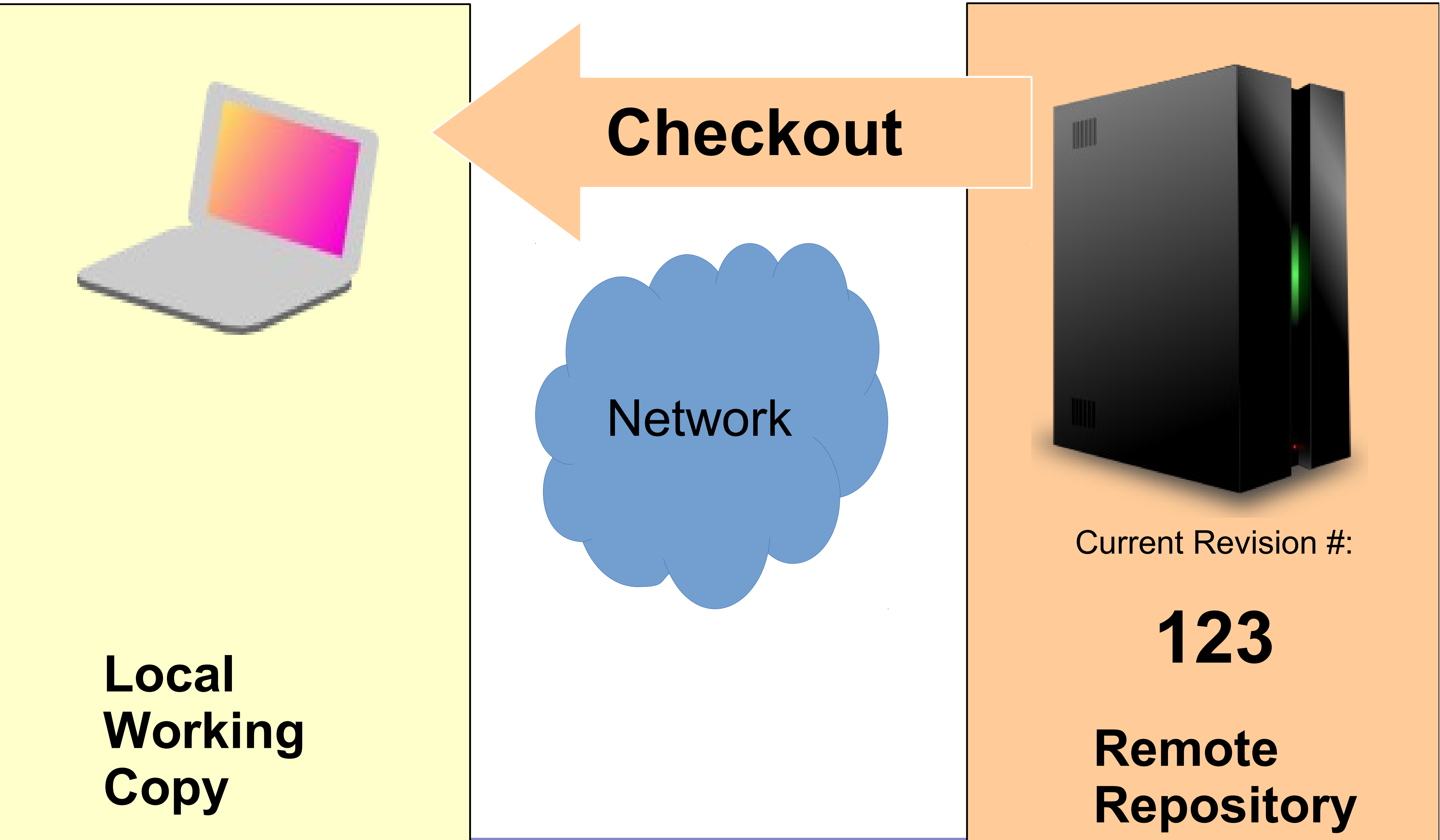
Concepts of version systems

- **Working copy**: the latest version of a set of files that you want to work on. This is usually **local** to your machine.
- **Revisions**: every “version” of one or more files gets a **revision tag**. This can be a number, a label, a string. Usually is increasing numbers. It somewhat identifies the moment in time when these files were “accepted” as good for the rest of the project.
  For this reason these systems are also known as
  Revision Systems
- **Commit**: the action assigning a revision number to the changes made in the working copy.
  The meaning is: I like the changes I did to these files, I accept them.
  It usually involves adding the files to a revision control **database**.
- **Checkout**: the action of retrieving a revision into a working copy.

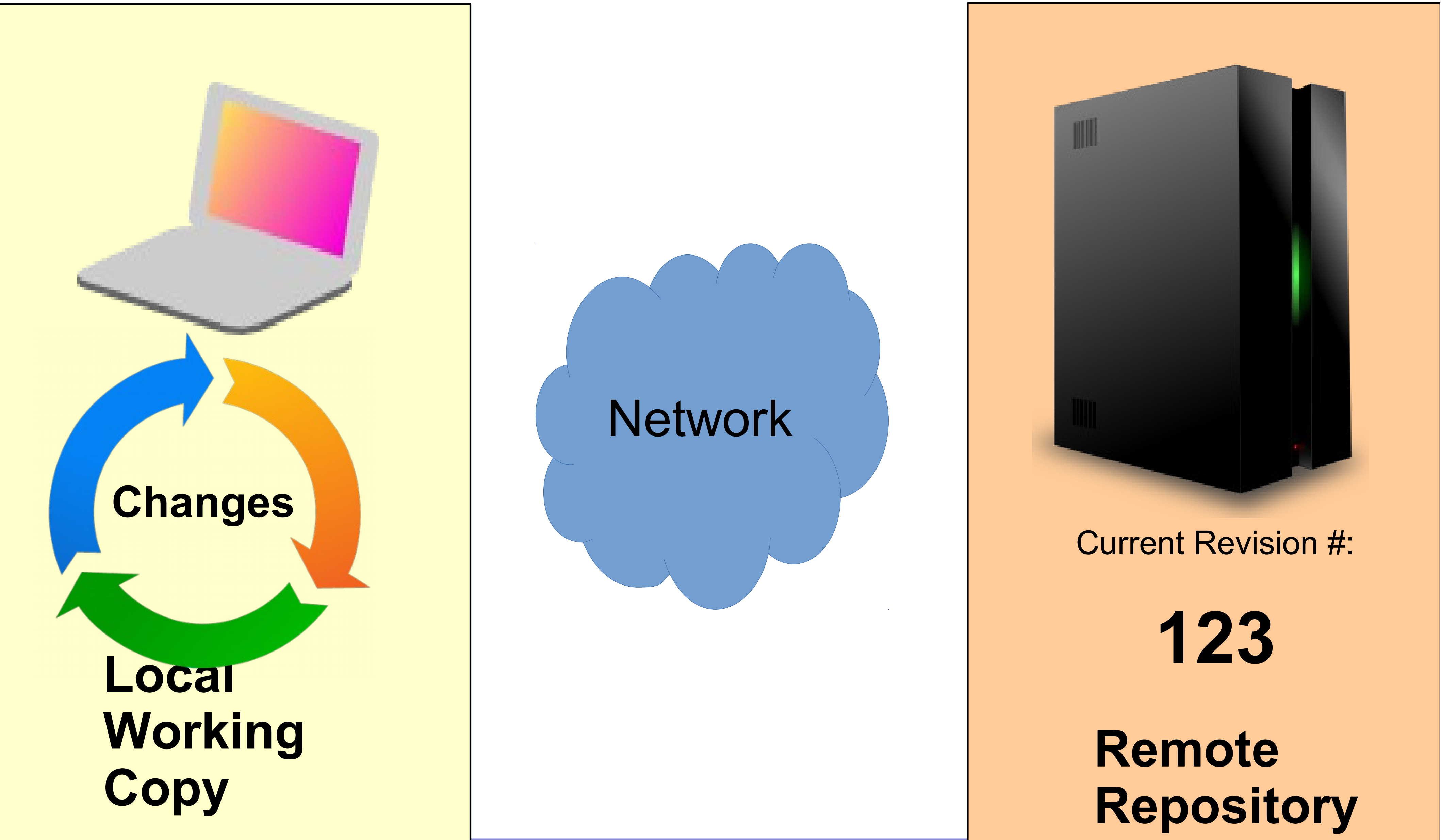
Concepts of version systems



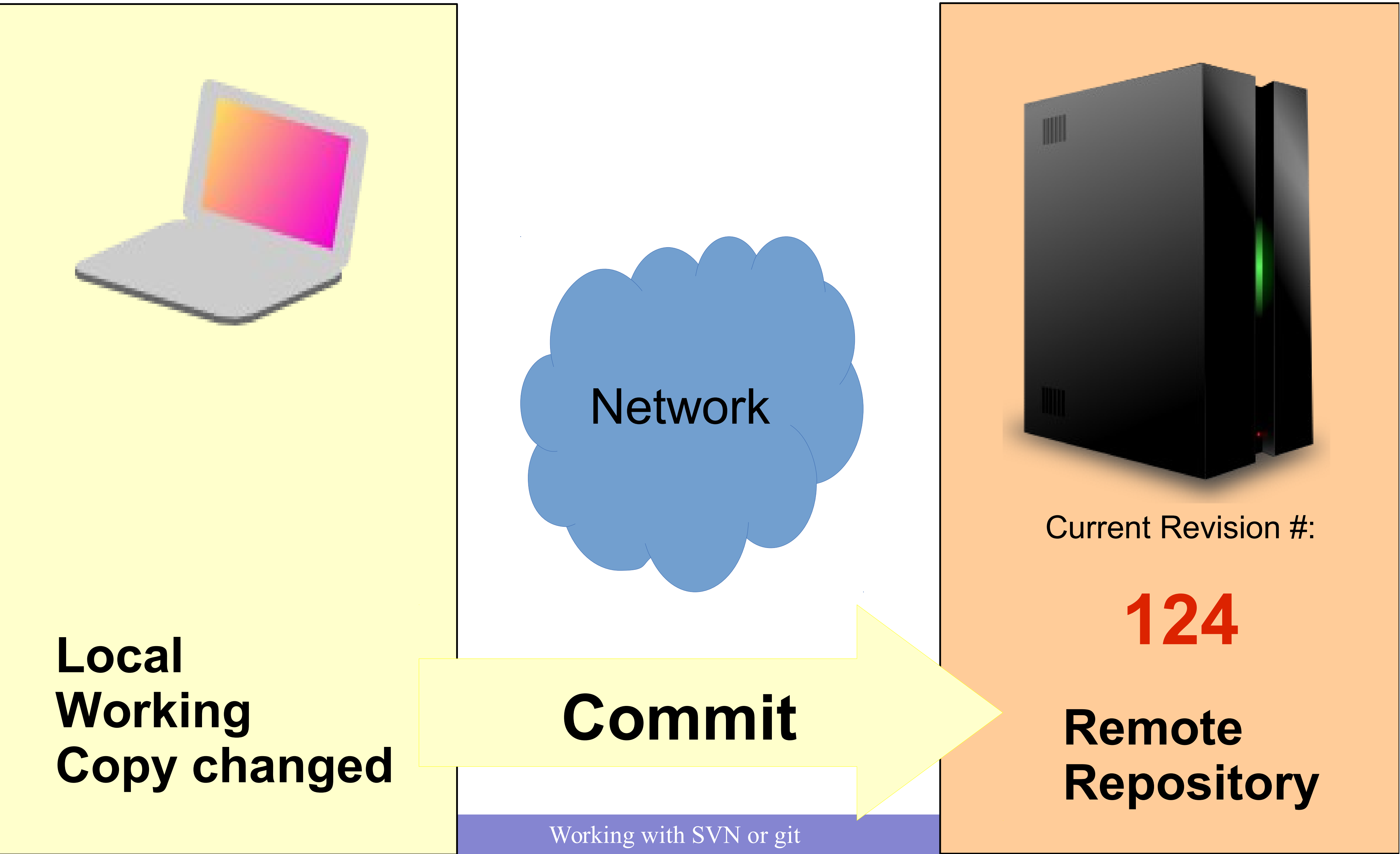
1. Checkout existing code from repo



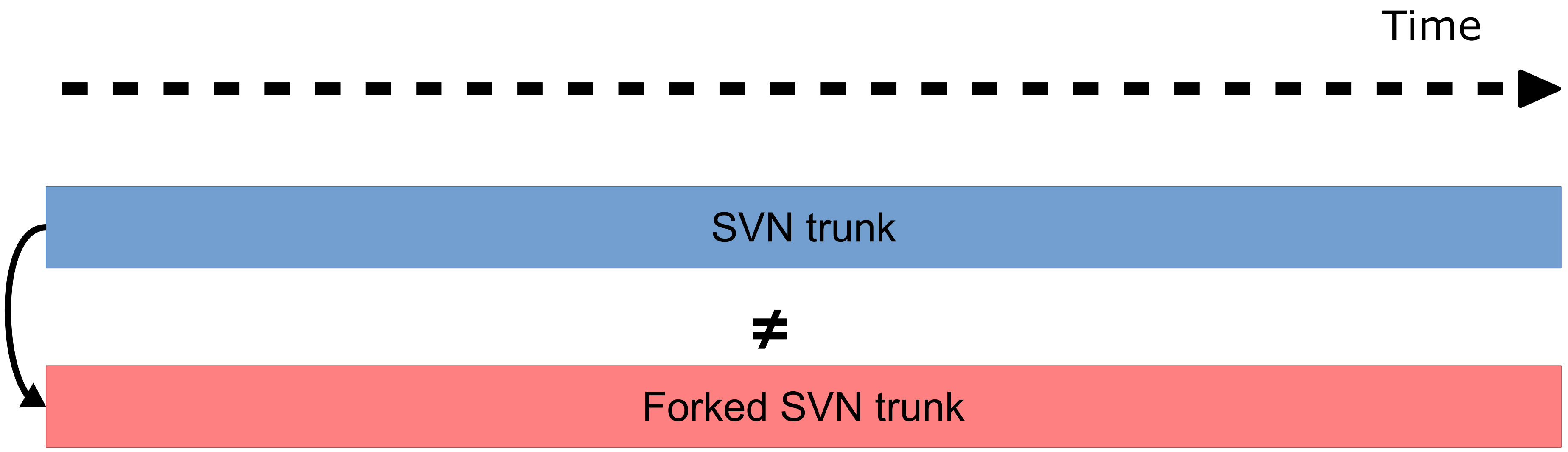
2. Make changes in the working copy



3. Commit a new version/revision

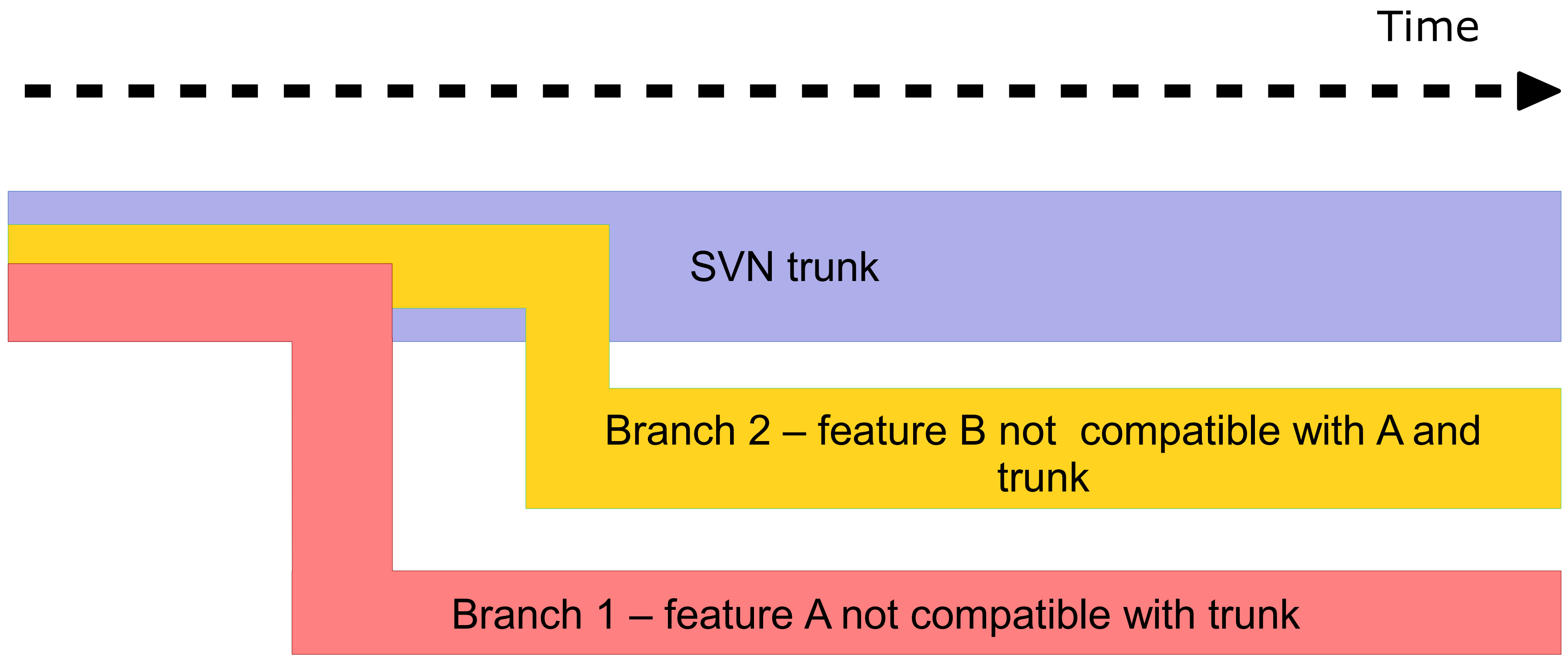


Fork



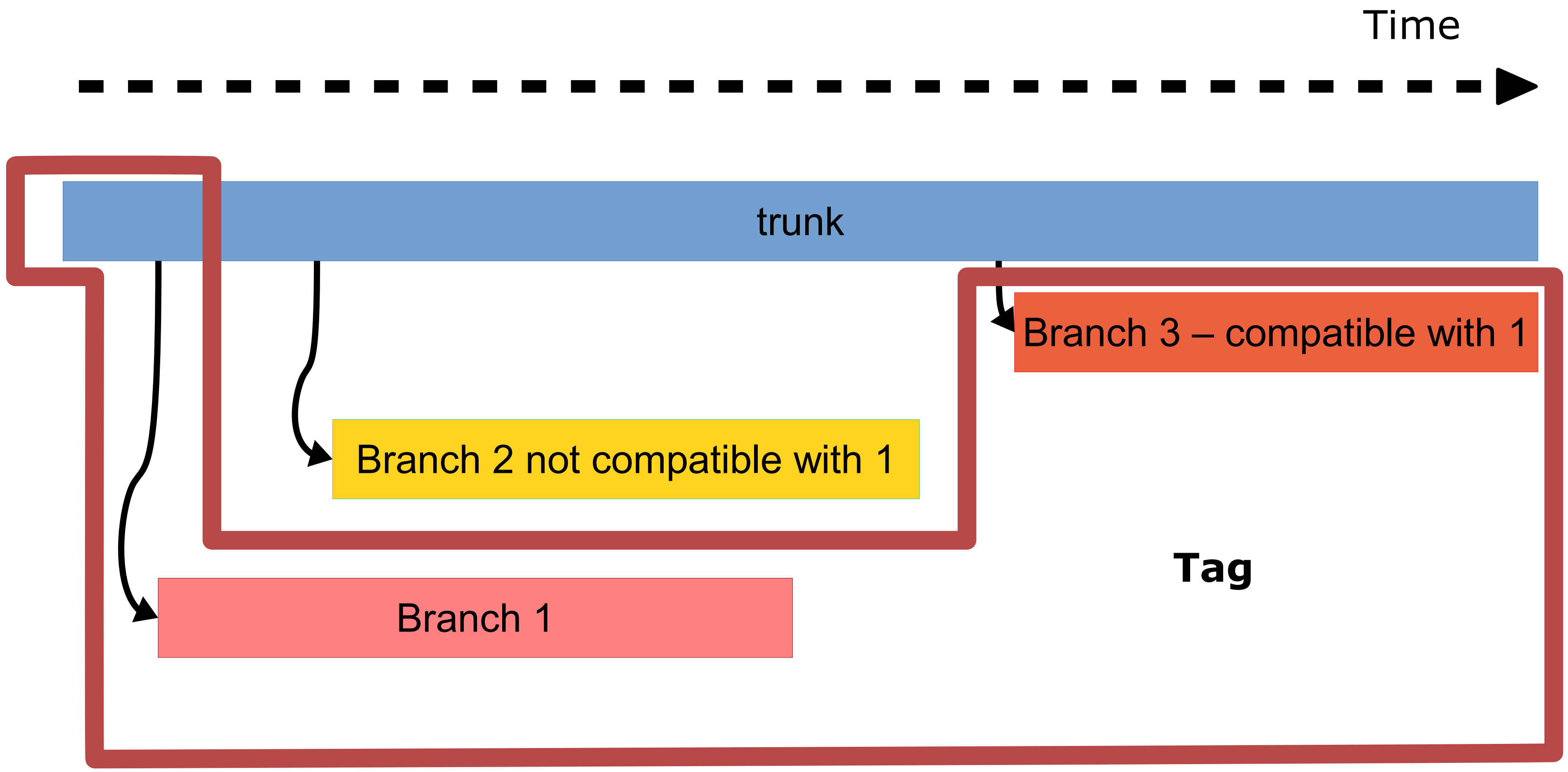
To copy a whole trunk into another working copy, to create a completely different program from the existing one.

Branch


Time
SVN trunk
Branch 2 – feature B not compatible with A and trunk
Branch 1 – feature A not compatible with trunk

To copy a whole trunk into another folder to add some features or functionalities that are not compatible with the original working copy

Tagging



Tagging: To copy a selected subset of the code in the working copy for it to be part of a specific release version of the software.

- **Release**: the copy of a working copy of a specific version of a software when made publicly accessible to users.

Version systems: products and features

Product	staging	Local commit	diff	Fork/branch management	Distributed/ Collaborative	Compatibility
CVS (Current Version Stable)	N	N	Y	Y	N	?
SVN (SubVersioN)	N	N	Y	N	N	?
Git	Y	Y	Y	Y	Y	SVN CVS

Preparing for the tutorial

- Install the SVN package via CLI:
  - `sudo apt-get install subversion`
- Create a folder in your home folder for sources:
  - `mdkir ~/svn/`
  - `cd ~/svn`

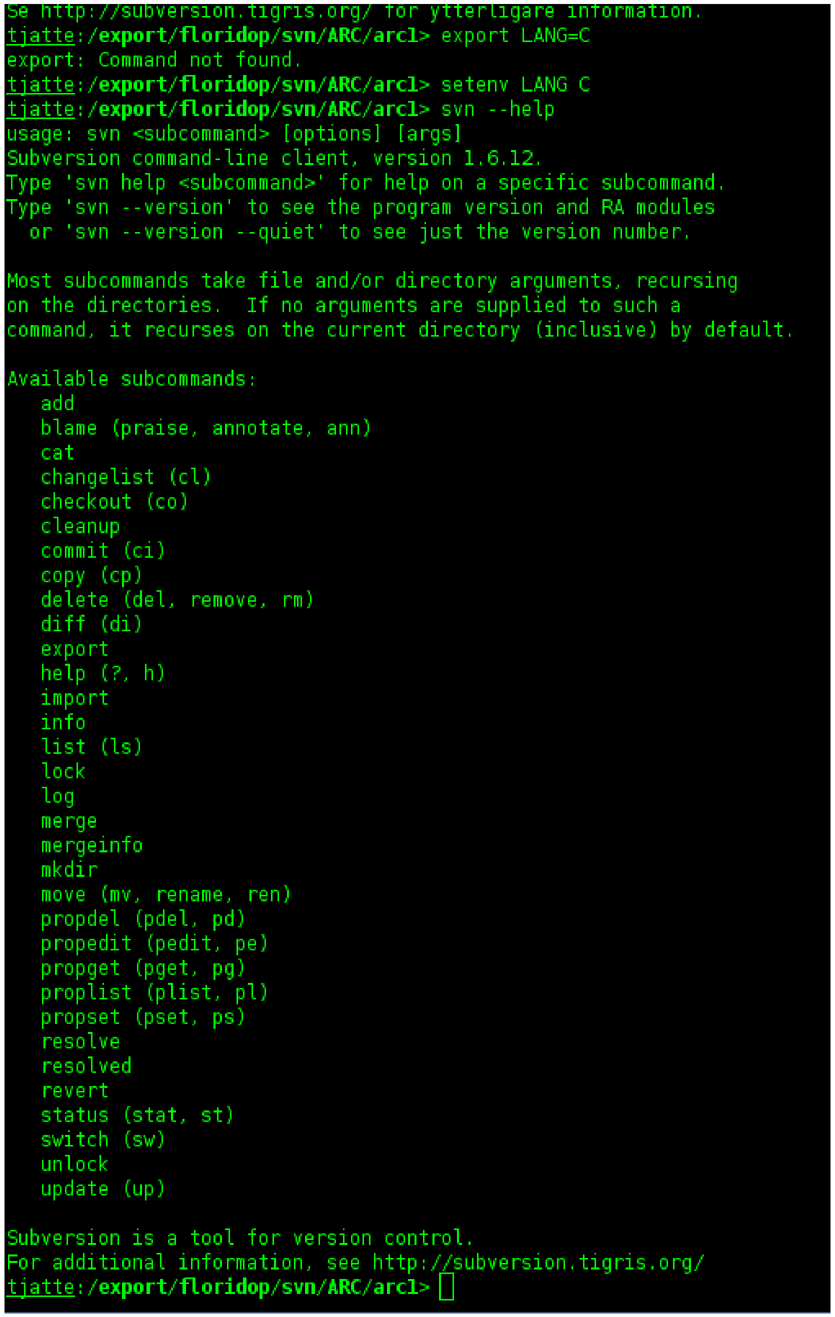
Subversion (SVN)

- Became the most widely used after CVS, but the two of them have orthogonal features
- **Stores the complete file at every revision**
- Has a database with the changes and revision logs
- Mainly **centralized**: a **server** keeps all the information, users checkout and commit. Every commit is assigned a new tag.
- Multiple users can access a repository.
- Tagging, branching, forking, merging are **done by hand** and are *based on conventions* on the folder names:
  - The **main** repository is stored in a folder called /**trunk**
  - **Branches** are stored in /**branches**
  - **Tags** are stored in **/tags**

SVN tutorial outline

- Checkout from a repository
- Add files to the working copy
- Commit changes to a repository
- Check changes
  - Diffing
  - Reverting
  - Merging
  - Resolution of conflicts
- Patching
- How to use it for your own code
- Graphical clients

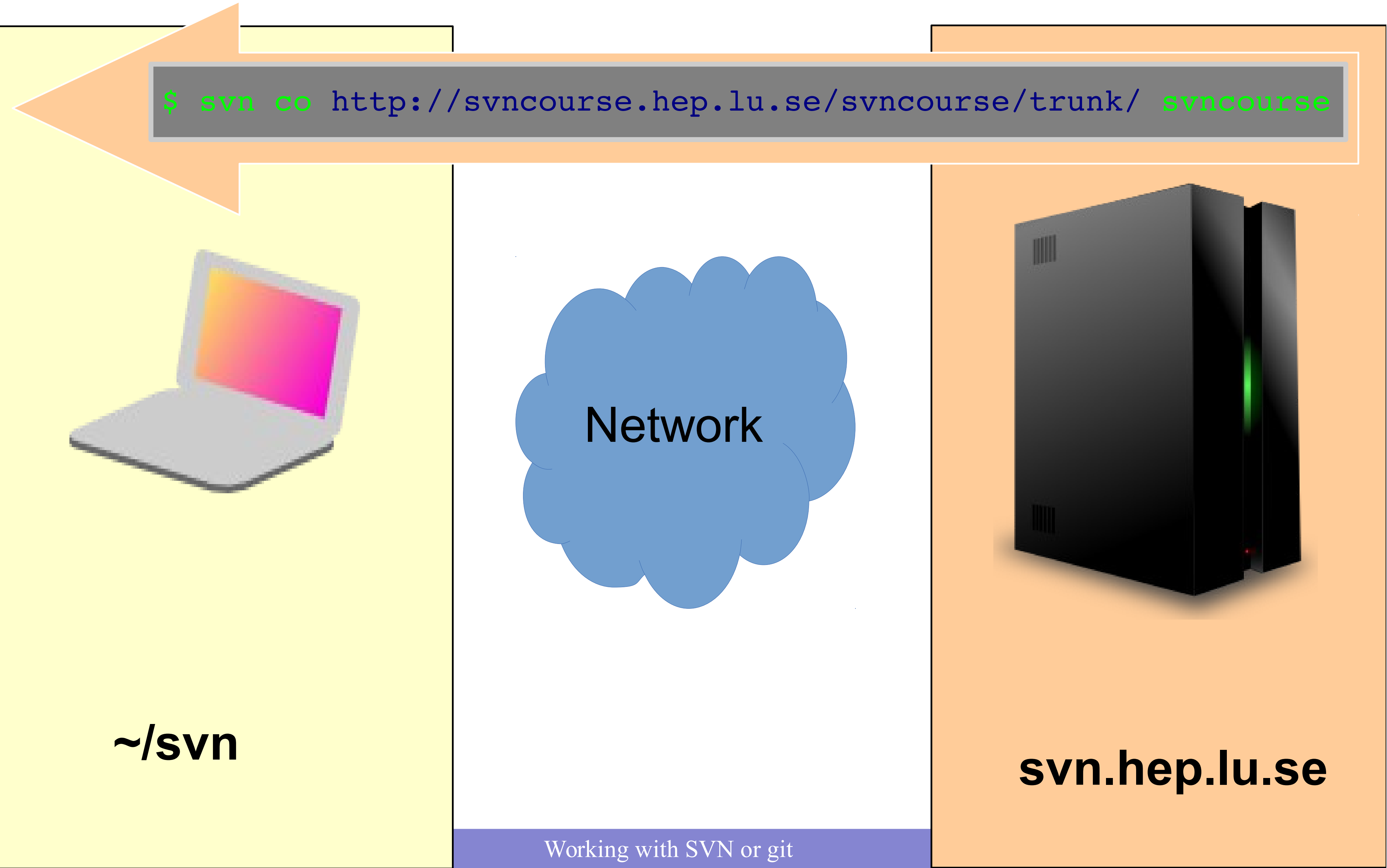
What commands are available?



```
Se http://subversion.tigris.org/ for ytterligare information.
tjatte:/export/floridop/svn/ARC/arcl> export LANG=C
export: Command not found.
tjatte:/export/floridop/svn/ARC/arcl> setenv LANG C
tjatte:/export/floridop/svn/ARC/arcl> svn --help
usage: svn <subcommand> [options] [args]
Subversion command-line client, version 1.6.12.
Type 'svn help <subcommand>' for help on a specific subcommand.
Type 'svn --version' to see the program version and RA modules
  or 'svn --version --quiet' to see just the version number.

Most subcommands take file and/or directory arguments, recursing
on the directories.  If no arguments are supplied to such a
command, it recurses on the current directory (inclusive) by default.

Available subcommands:
   add
   blame (praise, annotate, ann)
   cat
   changelist (cl)
   checkout (co)
   cleanup
   commit (ci)
   copy (cp)
   delete (del, remove, rm)
   diff (di)
   export
   help (?, h)
   import
   info
   list (ls)
   lock
   log
   merge
   mergeinfo
   mkdir
   move (mv, rename, ren)
   propdel (pdel, pd)
   propedit (pedit, pe)
   propget (pget, pg)
   proplist (plist, pl)
   propset (pset, ps)
   resolve
   resolved
   revert
   status (stat, st)
   switch (sw)
   unlock
   update (up)

Subversion is a tool for version control.
For additional information, see http://subversion.tigris.org/
tjatte:/export/floridop/svn/ARC/arcl>
```

```
$ svn --help
$ man svn
```

SVN checkout



```
$ svn co http://svncourse.hep.lu.se/svncourse/trunk/ svncourse
```

SVN checkout

```
> svn co http://svncourse.hep.lu.se/svncourse/trunk svncourse
Checked out revision 0.
```

- **`svn`** : the *subversion* command
- **`co`** : a shorthand for `checkout`
- **`http://svncourse.hep.lu.se/svncourse/trunk`**
  The name of the remote repository we want to sync with, and we take the upstream or main branch, trunk
- **`svncourse`**
  The local folder that will be created upon checkout
- **Revision**: a number assigned to a defined version of the code, that gets incremented at every **commit**.

Inspect the working copy

```
> cd svncourse
> ls -ltrah
total 16K
drwx--x--x 3 pflorido hep 4,0K 28 nov 16.50 ..
-rwx--x--x 1 pflorido hep 1,1K 28 nov 17.55 asciifun.py
drwx--x--x 6 pflorido hep 4,0K 28 nov 17.59 .svn
drwx--x--x 3 pflorido hep 4,0K 28 nov 18.01 .
```

- The .svn folder hosts the .svn database
- ! you should usually NOT touch this folder.

```
> svn info
Path: .
URL: http://svncourse.hep.lu.se/svncourse
Repository Root: http://svncourse.hep.lu.se/svncourse
Repository UUID: 3f457e7b-9635-49f8-9b60-cec4875accfe
Revision: 1
Node Kind: directory
Schedule: normal
Last Changed Author: courseuser
Last Changed Rev: 1
Last Changed Date: 2014-11-28 17:59:27 +0100 (fre, 28 nov 2014)
```

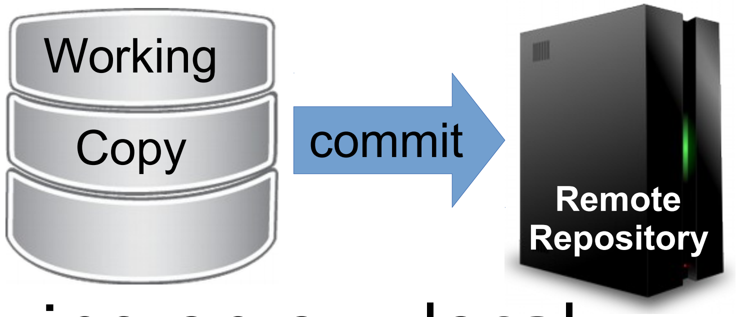
A side note about the example code in the repository

- The sample code uses some additional software and libraries that has nothing to do with SVN. To make the code work, you need to install these libraries manually (if they are not already in the virtual machine!) with the command:
  - `sudo apt-get install python-pyfiglet figlet`

Add files

- Create a copy of asciifun.py, save it with your name. Example: florido.py
- Edit the file and change the output text with your name.
- Run
  ```
  svn status
  ```
  What happens?
- An svn file can be in different statuses: use
  ```
  svn help status
  ```
  to discover them.
- The file we just created is not yet in the working copy database. We must add it with
  `svn add florido.py` (use your name here)
- Check `svn status` now. What happens?

Commit



- Up to now, the files are only staying on our local disk, in the working copy. But we want to share them, hence save them back on a remote repository!
- Run

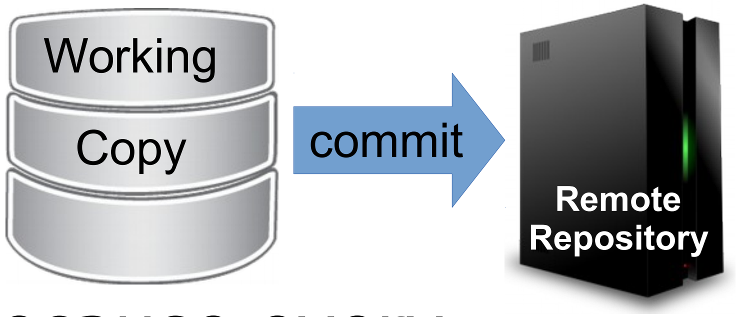
  ```
  svn commit --username=florido
  ```

  Using your first name.
  When asked, type the password (case sensitive):

ask the teacher

```
Password for ©florido©:
```

Commit



- A file editor will pop up. This is because every commit generates a **log**.
- A committer is requested to **describe the changes made** on the code and the effect it might have on the rest of the codebase.
- Once you save the file, the comment and the changes will be sent to the remote repository.

• Tip: the file editor can be changed.
For example, to use geany, execute:

```
export SVN_editor=geany
```

```
Password for ©florido©:
Adding          florido.py
Transmitting file data .
Committed revision 3.
```

Commit – what happened?

- Run
  ```
  svn info
  ```
- Run
  ```
  svn info http://svncourse/hep.lu.se/svncourse
  ```
- Discuss the differences with the teacher.

Commit – what happened?

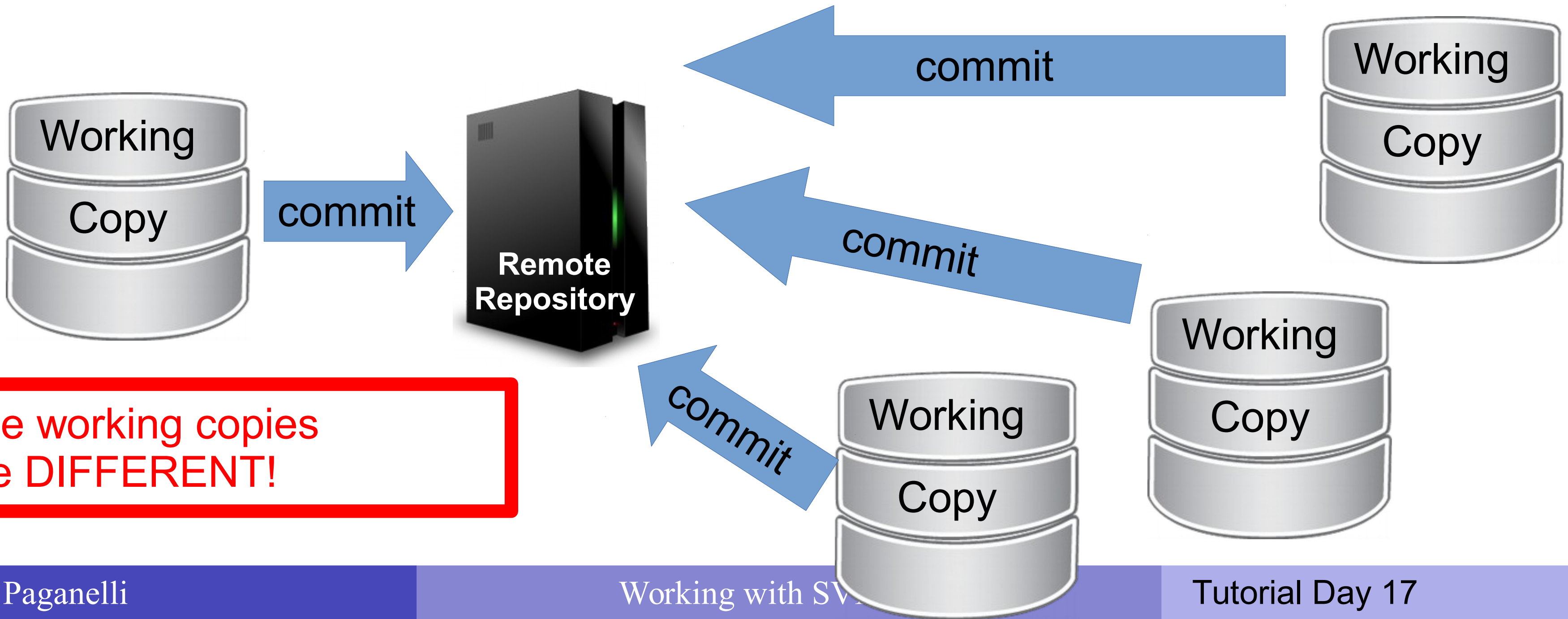
- Run

```
svn info
```

- Run

```
svn info http://svncourse/hep.lu.se/svncourse
```

- Discuss the differences with the teacher.



The working copies are DIFFERENT!

The commit log

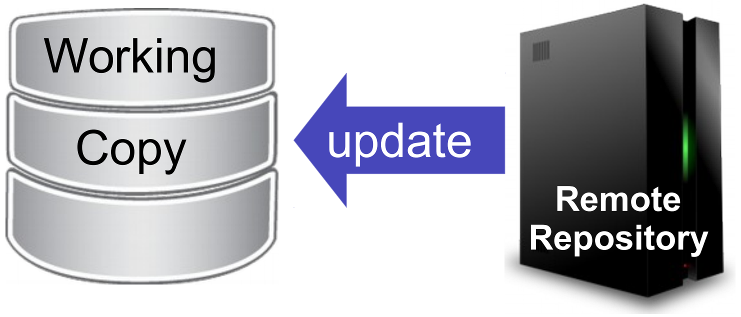
- Keeps track of the commits
- Run

  ```
  svn log -v
  ```

  to see it

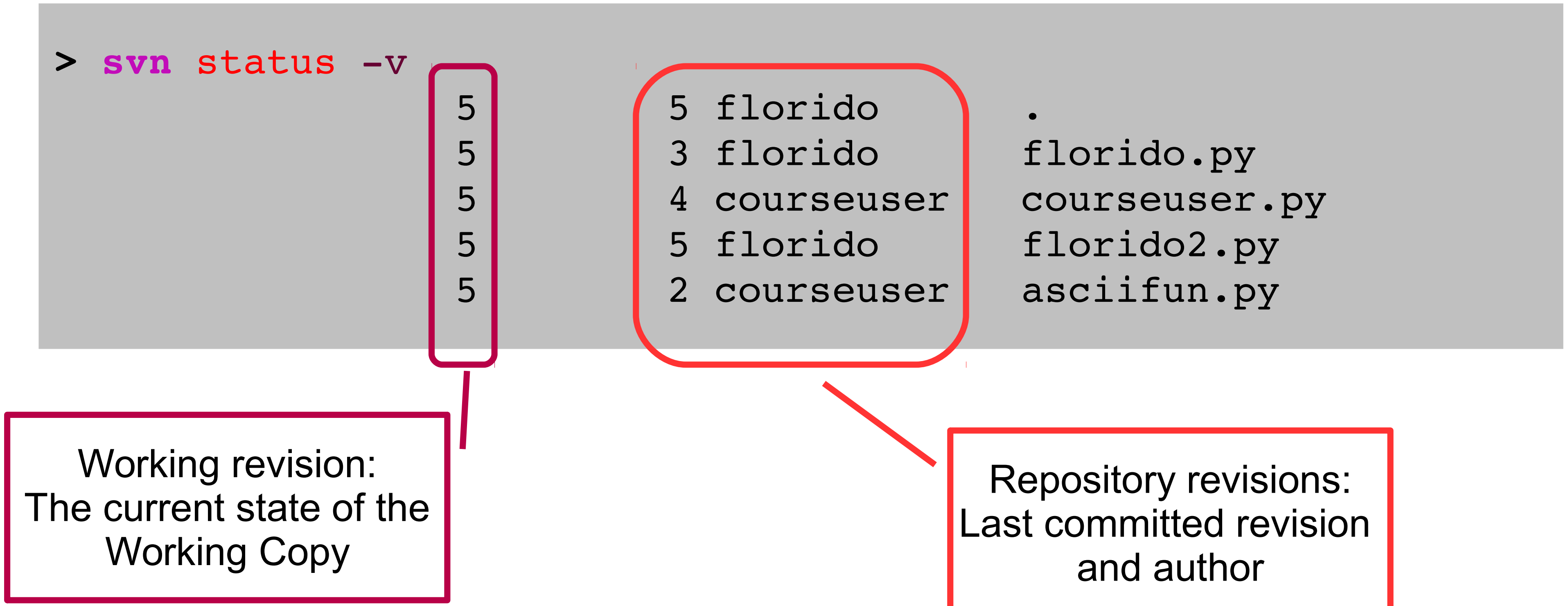
```
> svn log
------------------------------------------------------------------------
r2 | courseuser | 2014-12-01 09:28:51 +0100 | 2 lines
Changed paths:
   M /asciifun.py

Removed license comment

------------------------------------------------------------------------
r1 | courseuser | 2014-11-28 17:59:27 +0100 (fre, 28 nov 2014) | 5 lines
Changed paths:
   A /asciifun.py

First Commit

This includes the asciifun file, that prints
funny stuff on screen

------------------------------------------------------------------------
```

Update



- We need to sync the status of the remote repository with our local working copies. In this way we will get each other's contributions.
- Run
  ```
  svn update
  ```
- Run
  ```
  svn info
  ```
- Run
  ```
  svn status -v
  ```

Revision numbers explained



```
> svn status -v
                  5          5 florido        .
                  5          3 florido        florido.py
                  5          4 courseuser     courseuser.py
                  5          5 florido        florido2.py
                  5          2 courseuser     asciifun.py
```

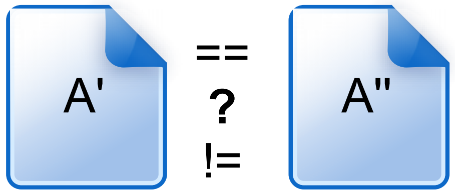
Working revision: The current state of the Working Copy
Repository revisions: Last committed revision and author

- To see the updates pending in the repository, use
  `svn status -vu`
- The asterisk * shows the updatable changes

Change code and commit

- **Best practice: before** changing anything, always do an `update`, so that you're sure you're working on the latest version of a file. Then you're safe to `commit`.
- Exercise:
  1. Update
  2. Do some changes in the file with your name
  3. run `svn status -uv`
  4. compare revisions
  5. Commit
  6. run `svn status -uv` again and discuss with the teacher.

Diffing



- Make some change in a file in your working copy.
- Run

  svn diff

```
> svn diff
Index: florido.py
===================================================================
--- florido.py   (revision 5)
+++ florido.py   (working copy)
@@ -8,7 +8,7 @@

 def main():
   f = Figlet(dir=©/usr/share/figlet/©,font=©pagga©)
-  print f.renderText(©It©s me, Florido!©)
+  print f.renderText(©I©m not saying it again, this is flo here©)
   return 0

 if __name__ == ©__main__©:
```

Reverting not committed changes

- Say that we are not happy with the changes we just made to a file and we want to go back to the repository version.
- Run

```
svn revert florido.py
svn diff
```

- **Careful! You will loose all the changes done and not committed!!!**

Reverting to a previous revision

- Say that we don't like the current revision state, and we want to roll back the code to a state of a different revision back in time.
- The main concept is:
  you never go back in the revision history.
  This is actually nice because in a collaborative environment, keeps track of who-did-what with no cheating allowed :)
- But in practice, this made cumbersome the way to revert to a previous revision. In fact, there are different methods to roll back a change. I will show you two.

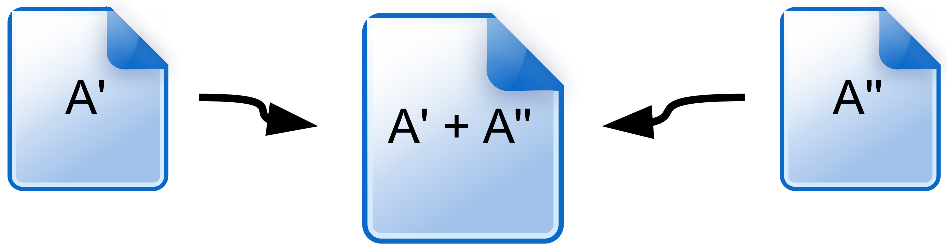
Revert to old revision: method 1 export

- SVN `export` is a command used to checkout a single file or a directory
- The easy way to rollback is to use it to export directly from and old revision into the working copy
- **NOTE**: you need to mention that there was a rollback in the commit comment, the system will not do for you.
- Exercise:
  - use export to roll back to one of the revisions of your file. Example:

    ```
    svn export -r 3 florido.py .
    ```

    will roll back `florido.py` to revision 3 in the folder `.` (current folder)
  - `svn diff`
  - `svn commit` the changes

Merging



- Suppose we have two versions of a document with different contents
- We want to make one out of two
- This is often referred as three-way-merge
- We need to choose which part of each document we want to keep
- There exist tools to do it, for example the excellent `meld`
- SVN can attempt to do merges for us:
  - If the merges are simple, i.e. the changed content of A' can be easily mised with that of the content of A''. For example, the documents differ a little but the changes in each document are not overlapping.
  - If we provide it with some hint on how to do the merges
  - If the above fail, it will ask us to do the merge manually, for example using `meld`
- The most frequent case of merge is in case of conflicts, we'll see it later!

Revert to old revision: method 2 reverse merge

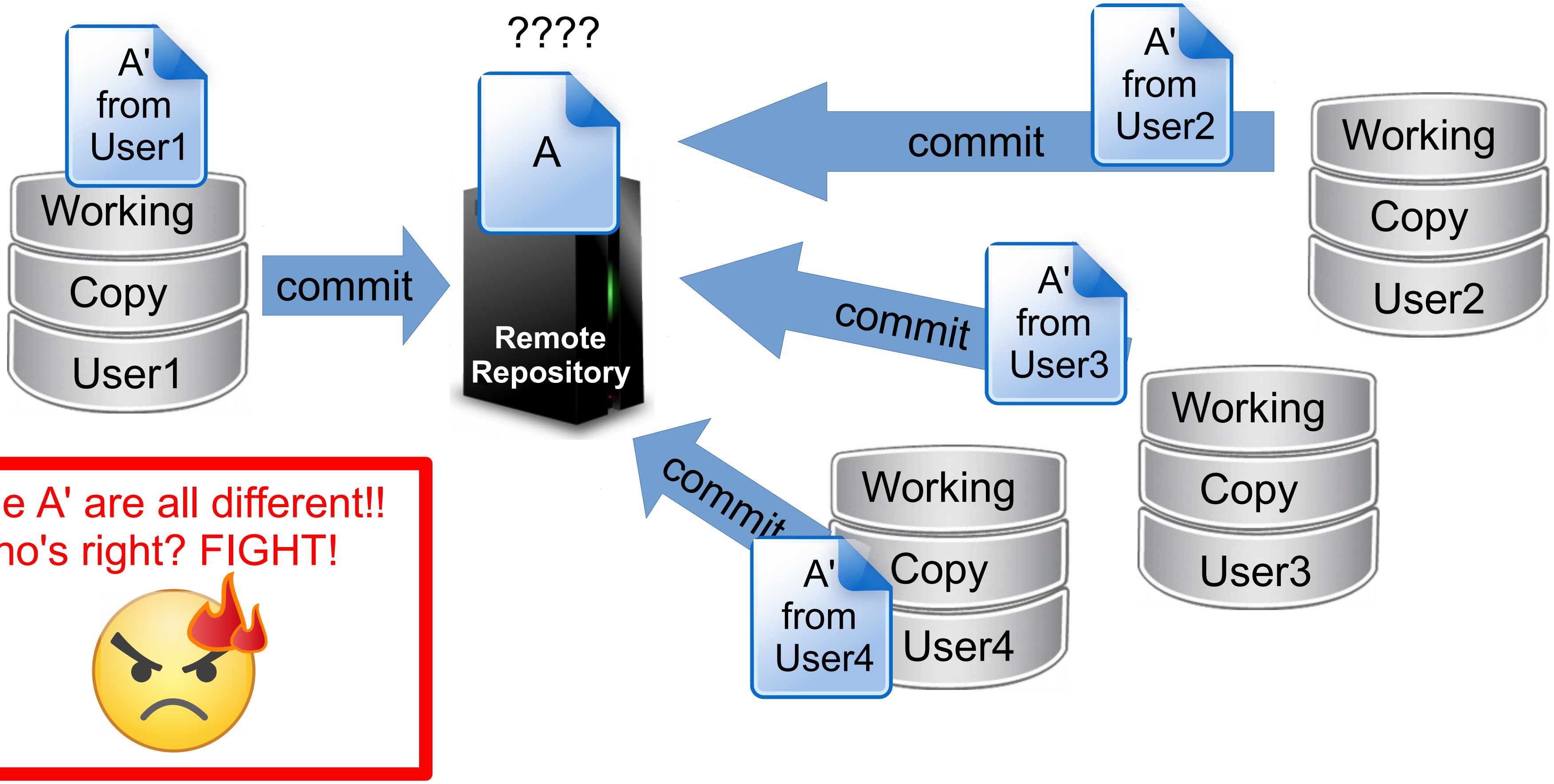
- **Reverse merge** is the name that SVN uses to represent the attempt to merge a document with a previous revision of the same document.
- Let's rollback one of our files to a previous revision:

```
svn merge -r HEAD:3 florido.py
```

- This will NOT change the file revision. Will just **copy** the content of the file at revision 3 into the latest (HEAD) revision. You can check with `svn diff` and `svn status -v`.

Conflicts

- A conflict happens when somebody edits a file that somebody else edited and committed before, and tries to commit without UPDATING.



- This usually happens when everybody is editing the same file.
  This is the reason why in big projects files are partitioned among programmers so that they don't write over each other.

Let's generate a conflict!

- Exercise: open and add some code (whatever!) to asciifun.py
- It can be:
  - Changing the font type, list of fonts as in `ls /usr/share/figlet/*.flf`
  - Changing the sentence
  - Adding another print...
  - Adding a for loop...
- All commit! The first to commit will be the winner :)

Handling a conflict

- The first to commit will set the new revision.
- If you try to commit now, SVN will complain that your version is not up to date with the repository
- If you try to update, SVN will notice that the file you changed has been already changed on the repository: this is called a **conflict.**
- Depending on the complexity of the changes made, SVN may or may not try do do a merge for you. If it fails, it will ask you to resolve the conflict manually.

Conflicts resolution

- When a conflict is found, SVN shows several options to resolve it:

```
> svn up
Conflict discovered in ©asciifun.py©.
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: s

  (e)  edit             - change merged file in an editor
  (df) diff-full        - show all changes made to merged file
  (r)  resolved         - accept merged version of file

  (dc) display-conflict - show all conflicts (ignoring merged version)
  (mc) mine-conflict    - accept my version for all conflicts (same)
  (tc) theirs-conflict  - accept their version for all conflicts (same)

  (mf) mine-full        - accept my version of entire file (even non-conflicts)
  (tf) theirs-full      - accept their version of entire file (same)

  (p)  postpone         - mark the conflict to be resolved later
  (l)  launch           - launch external tool to resolve conflict
  (s)  show all         - show this list
```

Conflicts resolution - diff

- Let's use df to see what the changes are:

```
        (s) show all options: df
--- .svn/text-base/asciifun.py.svn-base   ons dec  3 11:53:47 2014
+++ .svn/tmp/asciifun.py.2.tmp    ons dec  3 11:55:36 2014
@@ -7,7 +7,11 @@
 from pyfiglet import Figlet

 def main():
-    f = Figlet(font=©pagga©)
+<<<<<<< .mine
+    f = Figlet(font=©slant©)
+=======
+    f = Figlet(font=©futura©)
+>>>>>>> .r13
    print f.renderText(©This text is awesome! :D©)
        for x in range (0, 3):
        print f.renderText("the time is %d" % (x))
Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options:
```

Conflicts resolution - diff

- Let's use df to see what the changes are:

```
        (s) show all options: df
--- .svn/text-base/asciifun.py.svn-base   ons dec  3 11:53:47 2014
+++ .svn/tmp/asciifun.py.2.tmp    ons dec  3 11:55:36 2014
@@ -7,7 +7,11 @@
 from pyfiglet import Figlet

 def main():
-    f = Figlet(font=©pagga©)
+<<<<<<< .mine
+    f = Figlet(font=©slant©)
+=======
+    f = Figlet(font=©futura©)
+>>>>>>> .r13
    print f.renderText(©This text is awesome! :D©)
         for x in range (0, 3):
        print f.renderText("the time is %d" % (x))
Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options:
```

mine : The changes in the working copy

Divider between the two changes

r13 : The changes in the remote repos

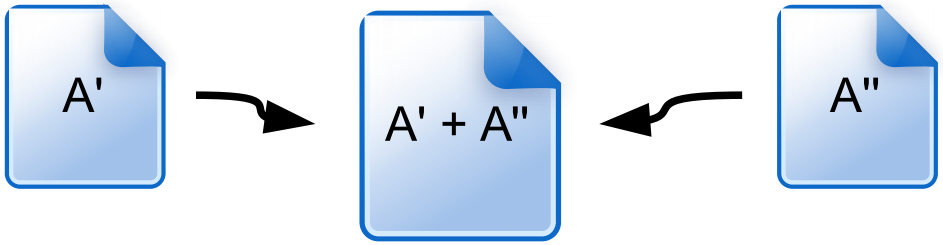
Conflicts resolution - diff

- **mine-conflict**: select my changes and resolve conflict
- **theirs-conflict**: select the repository changes and resolve the conflict
- **edit**: open an editor and solve the conflict manually
- **resolve**: leave the file with this funny structure and resolve the conflict
- **merge**: use SVN builtin tool to merge
- **launch**: use external tool to merge
- **postpone**: leave the file with the funny structure, but do NOT resolve the conflict!

Conflicts resolution - diff

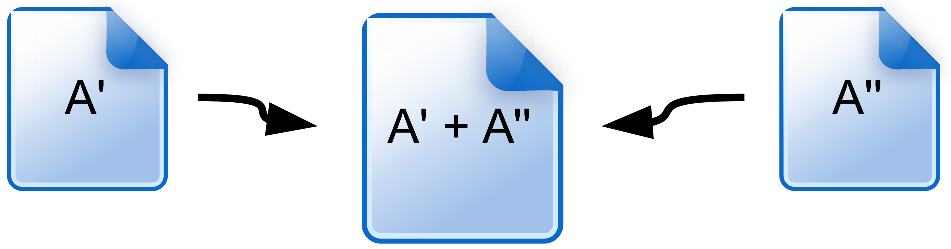
- Exercise:
  - Look at merge option. Do not merge! Go back with abort (a)
  - Try the launch option. What happens?
- **Let's go postpone**: we will resolve the conflict using `meld`
- List the contents of the SVN directory. What happened?

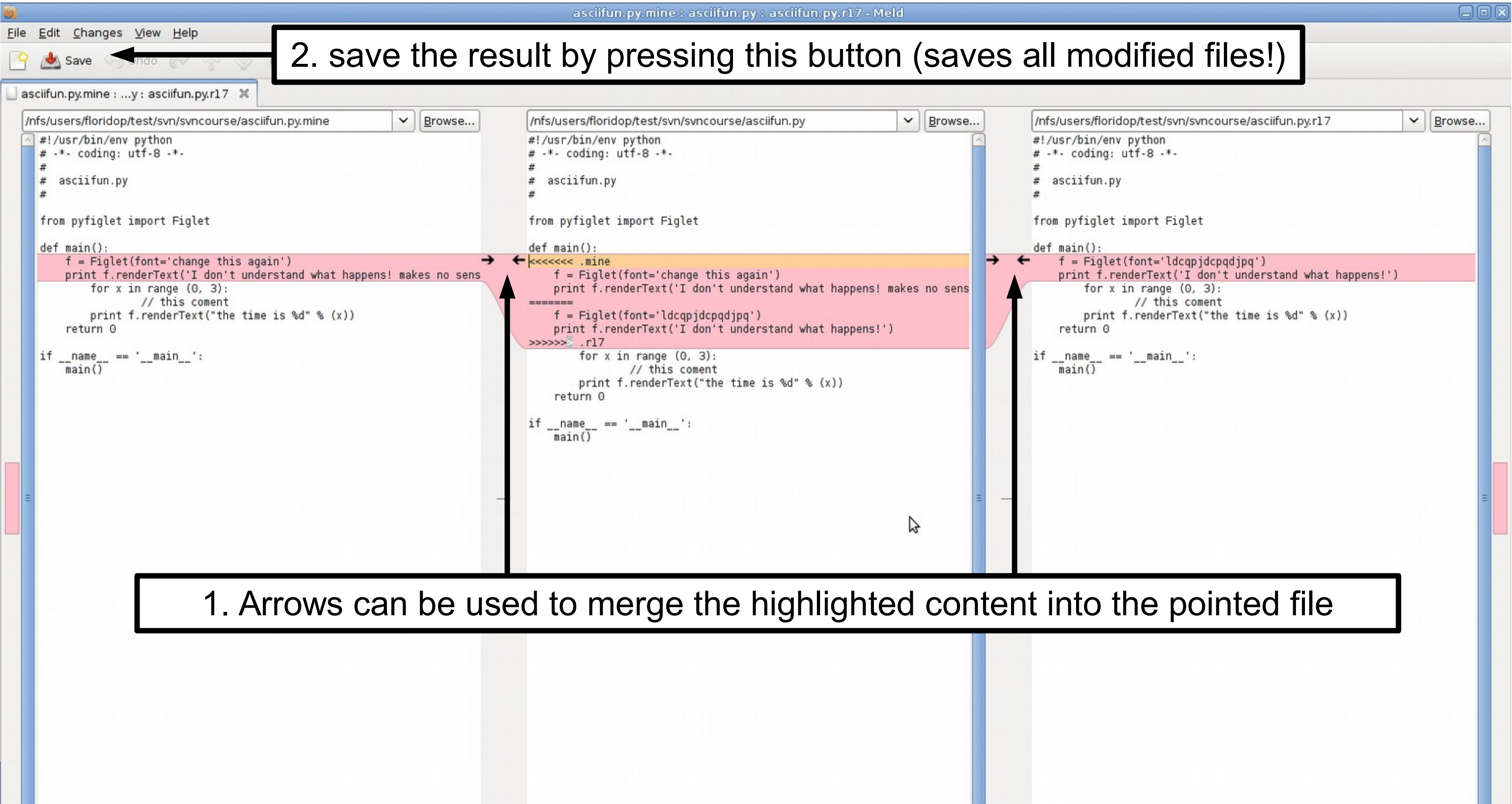
Merging with `meld`



- We now have three version of a document we want to merge into one.
- Meld command line syntax is as follows:
  ```
  meld file1 file2 file3
  ```
- The best is to use it this way:
  ```
  meld source1 destination source2
  ```
- That means, we want to merge the contents of the files `source1` and `source2` into **`destination`.**
- In our case:
  ```
  meld asciifun.py.mine asciifun.py asciifun.r16
  ```
  where `r16` is revision number that conflicts, written by SVN when we chose **postpone**.
- Run it!

Merging with `meld`




asciifun.py.mine : asciifun.py : asciifun.py.r17 - Meld
File Edit Changes View Help
Save
Undo
asciifun.py.mine : ...y : asciifun.py.r17
Browse...

2. save the result by pressing this button (saves all modified files!)

/nfs/users/floridop/test/svn/svncourse/asciifun.py.mine

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  asciifun.py
#

from pyfiglet import Figlet

def main():
    f = Figlet(font='change this again')
    print f.renderText('I don't understand what happens! makes no sens
        for x in range (0, 3):
            // this coment
        print f.renderText("the time is %d" % (x))
    return 0

if __name__ == '__main__':
    main()
```

/nfs/users/floridop/test/svn/svncourse/asciifun.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  asciifun.py
#

from pyfiglet import Figlet

def main():
<<<<<<< .mine
    f = Figlet(font='change this again')
    print f.renderText('I don't understand what happens! makes no sens
=======
    f = Figlet(font='ldcqpjdcpqdjpq')
    print f.renderText('I don't understand what happens!')
>>>>>>> .r17
        for x in range (0, 3):
            // this coment
        print f.renderText("the time is %d" % (x))
    return 0

if __name__ == '__main__':
    main()
```

/nfs/users/floridop/test/svn/svncourse/asciifun.py.r17

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  asciifun.py
#

from pyfiglet import Figlet

def main():
    f = Figlet(font='ldcqpjdcpqdjpq')
    print f.renderText('I don't understand what happens!')
        for x in range (0, 3):
            // this coment
        print f.renderText("the time is %d" % (x))
    return 0

if __name__ == '__main__':
    main()
```

1. Arrows can be used to merge the highlighted content into the pointed file

Conflicts resolution: resolved

- Once we're done with resolving the conflict, we can tell the SVN system to accept the resolution. This is done using the command

```
svn resolved asciifun.py
```

- After this, we're ready to commit.

```
> svn commit
svn: Commit failed (details follow):
svn: Aborting commit: ©/nfs/users/floridop/test/svn/svncourse/asciifun.py©
     remains in conflict
> svn resolved asciifun.py
Resolved conflicted state of ©asciifun.py©
> svn commit
```

Creating and applying patches

- A **patch** is a special file containing information on how to fix a certain problem.
  - It's called “patch” because its fixes can be applied on top of what already exist.
- In the computer world, a patch can be either a binary or a source file. We will not discuss binary patches, only source code patches.
- The format of a patch is similar to the diff format we've seen already.

Creating patches

- A way of creating a patch is to use the `svn diff` command.
- Say that we gave the code of asciifun.py file at revision3 to a friend, and we want to give the latest version.
  - The friend does not want to use SVN
  - He has very limited space to carry the new code around, for example on a usb pen. He just wants the newer parts.

Creating patches wit svn diff

- The syntax for the svn diff command is as follows:

```
svn diff -r asciifun.py@3 asciifun@HEAD
```

- This generates a patch file output. What we have to do is write the output to a file:

```
svn diff -r asciifun.py@3 asciifun@HEAD >
asciifun.py.20141212.patch
```

Applying patches with patch

- We're about to use a program called "patch", that does three way merge of different files given the patch file previously created.
- **ALWAYS READ THE CONTENTS OF A PATCH FILE BEFORE APPLYING IT**
  - You can never be sure it doesn't contain malicious code!!
- Let's restore revision 3 of asciitest.py to test the patch.
    - Create a folder in your home
      `mkdir ~/test/`
    - Export to that folder asciifun.py at revision 3 with `svn export` (check previous slides!)
    - copy the **`asciifun.py.20141212.patch`** patch file into the ~/test/ folder
    - `cd` into the test folder

Applying patches with patch

- Make sure that both the revision 3 asciifun.py file and the **`asciifun.py.20141212.patch`** files are in the `~/test/` folder.
- `cat` the content of asciifun.py
- Run the following:

```
patch -p0 -i asciifun.py.20141212.patch
```

  - **`-p0`**: go up of 0 directories (it does `cd ../` as many times as the indicated number)
  - **`-i asciifun.py.20141212.patch`**: use **`asciifun.py.20141212.patch`** as input file that contains instructions how to patch.
- `cat` the content of asciifun.py again. It changed!

Try this at home!

Or, How to benefit of revision control for your own code

- One does not necessarily need a remote repository. By installing subversion tools one gets also all the needed to create a repo himself.
- So if you get to do some coding in the future, create your own repository:

  ```
  svnadmin create ~/mysvnrepo
  ```

  It will create a directory `myrepo` that contains the database.
- Add the files you want to track/version/revise to the database:

  ```
  svn import /path/to/filestotrack/
  file:///home/username/mysvnrepo -m ªIntial import of filesº
  ```
- From now on you can checkout the repository using

  ```
  svn co file:///home/username/mysvnrepo /path/to/workingcopy/
  ```

  And work inside `/path/to/workingcopy/`

Graphical Clients

- Want to try a graphical client?
  - Minimal one: run
    ```
    rapidsvn
    ```
    - This one is available in Lubuntu repositories.
  - Feature-rich one:
    ```
    cd ~/Software
    cd smartsvn-8_6_2
    cd bin
    ./smartsvn.sh
    ```
    - This one is NOT available on Lubuntu repositories. You need to download it from the internet. http://www.wandisco.com/smartsvn/home
- A repository can also be equipped with cool network tools to share and visualize the changes, like TRAC. An example from NorduGrid SVN:
  - http://svn.nordugrid.org/trac/nordugrid/
  - Big example: Click here

References

- SVN Quick Reference Card:
  http://wiki.ssg.uab.edu/download/attachments/3080576/Subversion+Quick+Reference+Card.pdf?version=1
- The SVN Redbook
  http://svnbook.red-bean.com/
- Patching with SVN:
  https://ariejan.net/2007/07/03/how-to-create-and-apply-a-patch-with-subversion/

Pictures references

- https://openclipart.org/
- http://www.libreoffice.org/