Working with SVN

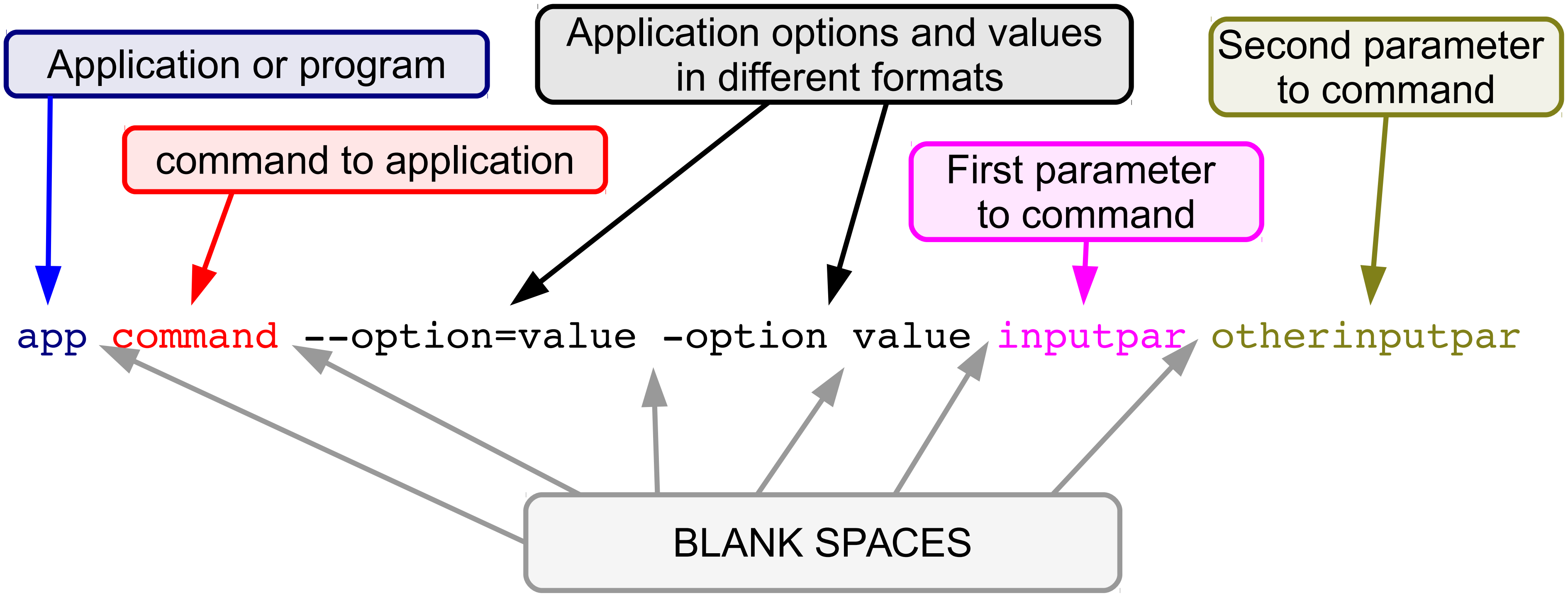
Florido Paganelli
Lund University
florido.paganelli@hep.lu.se

Outline

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

Notation

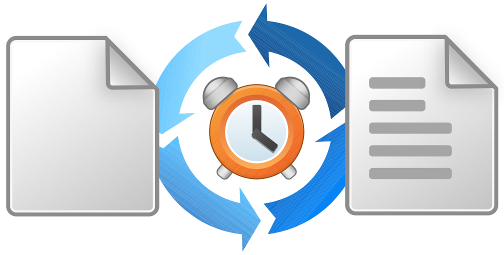
- I will be using the following color code for showing commands:


Application or program
command to application
Application options and values in different formats
First parameter to command
Second parameter to command
BLANK SPACES

```
app command --option=value -option value inputpar otherinputpar
```

Why version/revision systems?

- Say you wrote some piece of code.
- You discover a bug and you want to change it.
- You fix the bug, save the code. Try the program again and... it doesn't work!
- **What went wrong?** Would be nice if you could **compare** what you **changed**...
- **Solution:** make a backup copy <u>before every change</u>!
- Version systems make it easy to backup and compare **changes**



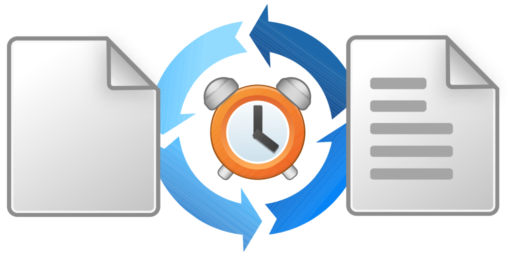
Why version/revision systems?

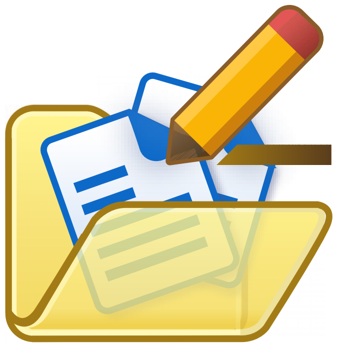
- If you do many changes, you might not remember what changes you made. Version systems allow you to attach a **comment** to the change.

- If you want to share your code with other developers, it's nice if everybody can see who changed what. Version systems allow you to **author** the changes so the others know what you're done. This helps **sharing** code.

Why version/revision systems?

- Summary:
  - **Backup** each change in your code
  - **Compare** different versions of your code
  - **Comment** and annotate each change
  - **Share** among developers





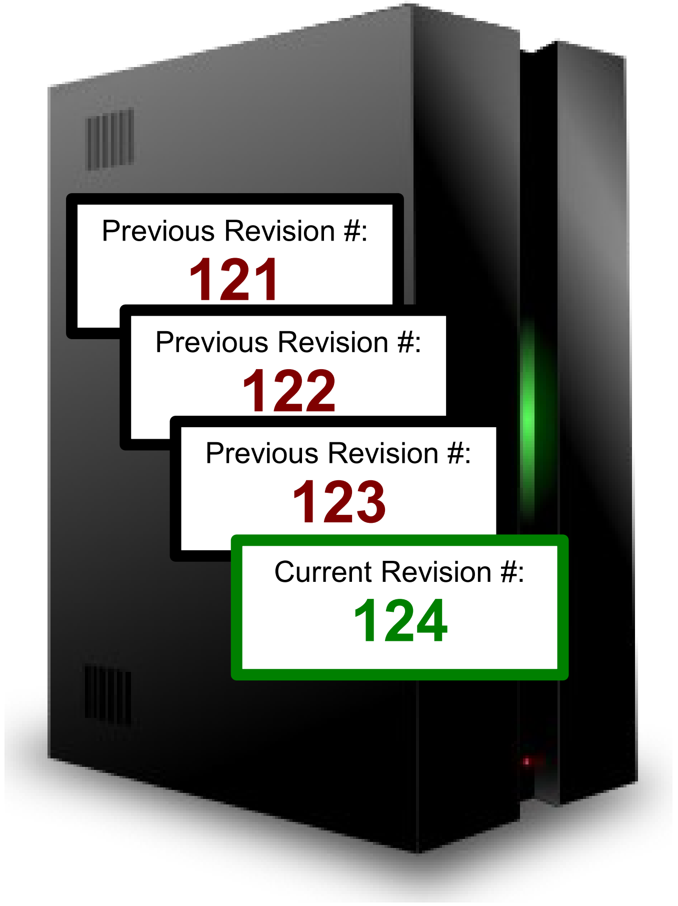
Concepts of version systems

- **Repository**: A database that contains the list of changes made. Can be on a **remote server** or even in a folder **local** to your machine.

- **Working copy**: the latest version of a set of files that you want to work on. This is usually **local** to your machine.

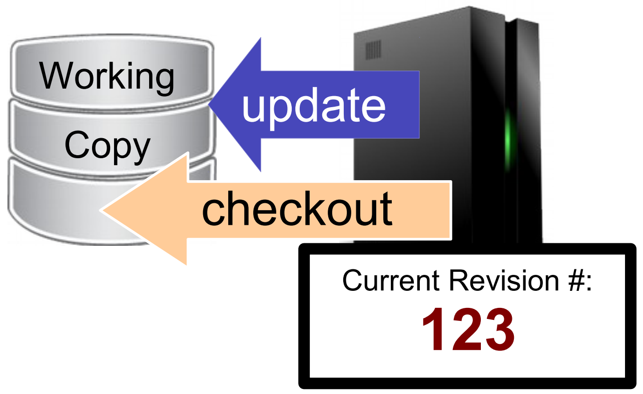
Concepts of version systems

- **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**

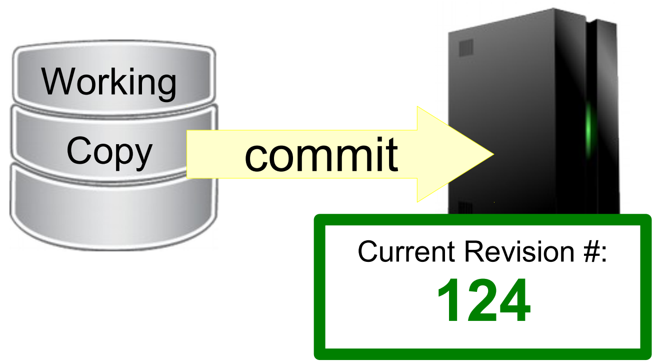


Concepts of version systems

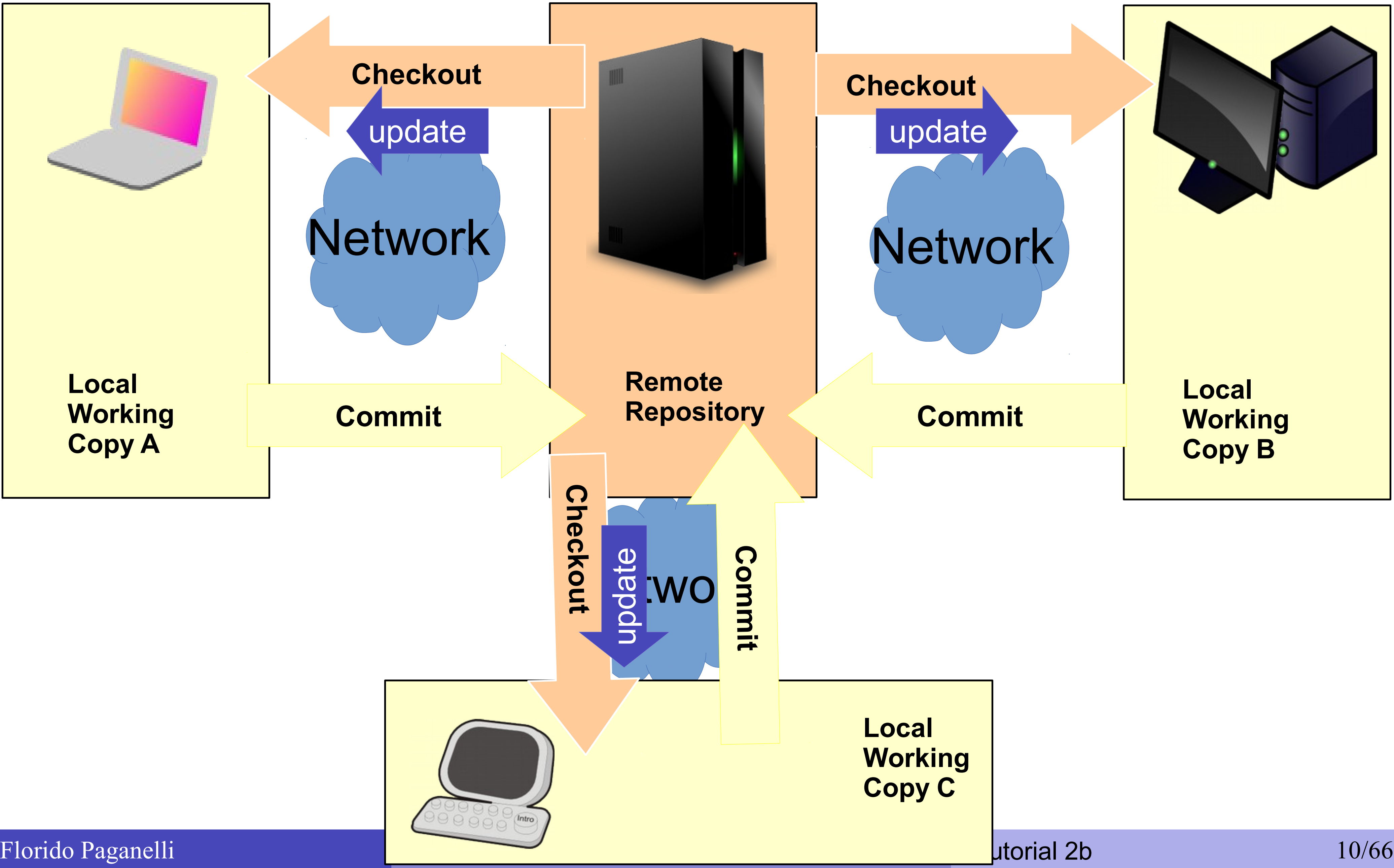
- **Checkout and update:** the actions of retrieving a revision into a working copy:
  - **Checkout** is used the first time to create a working copy.
  - **Update** is used to synch an existing working copy.



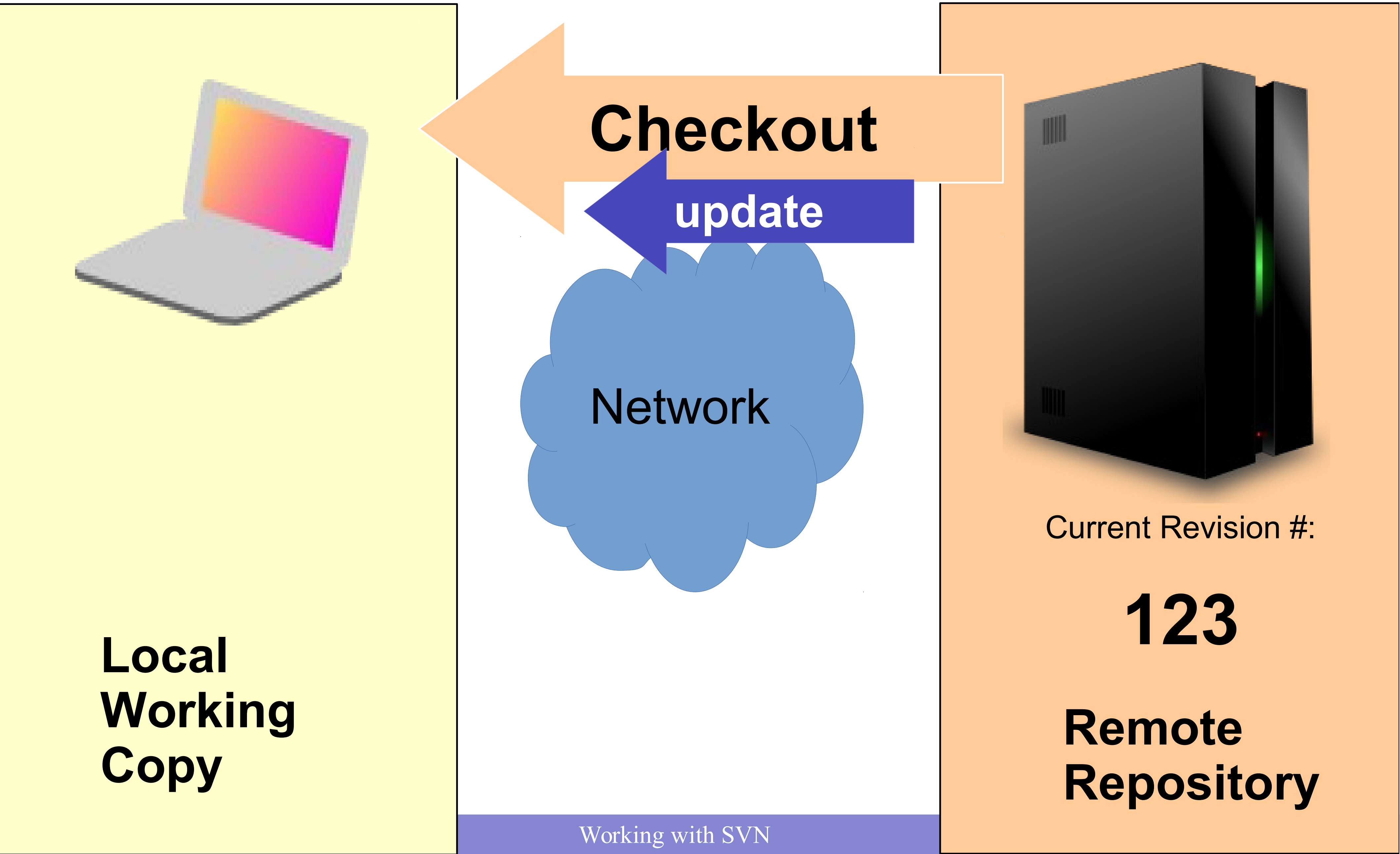
- **Commit**: the action assigning a revision number to the changes made in the working copy.
  The meaning is: I like the changes I did to these files, I accept them.
  It usually involves adding the files to a revision control **database**.



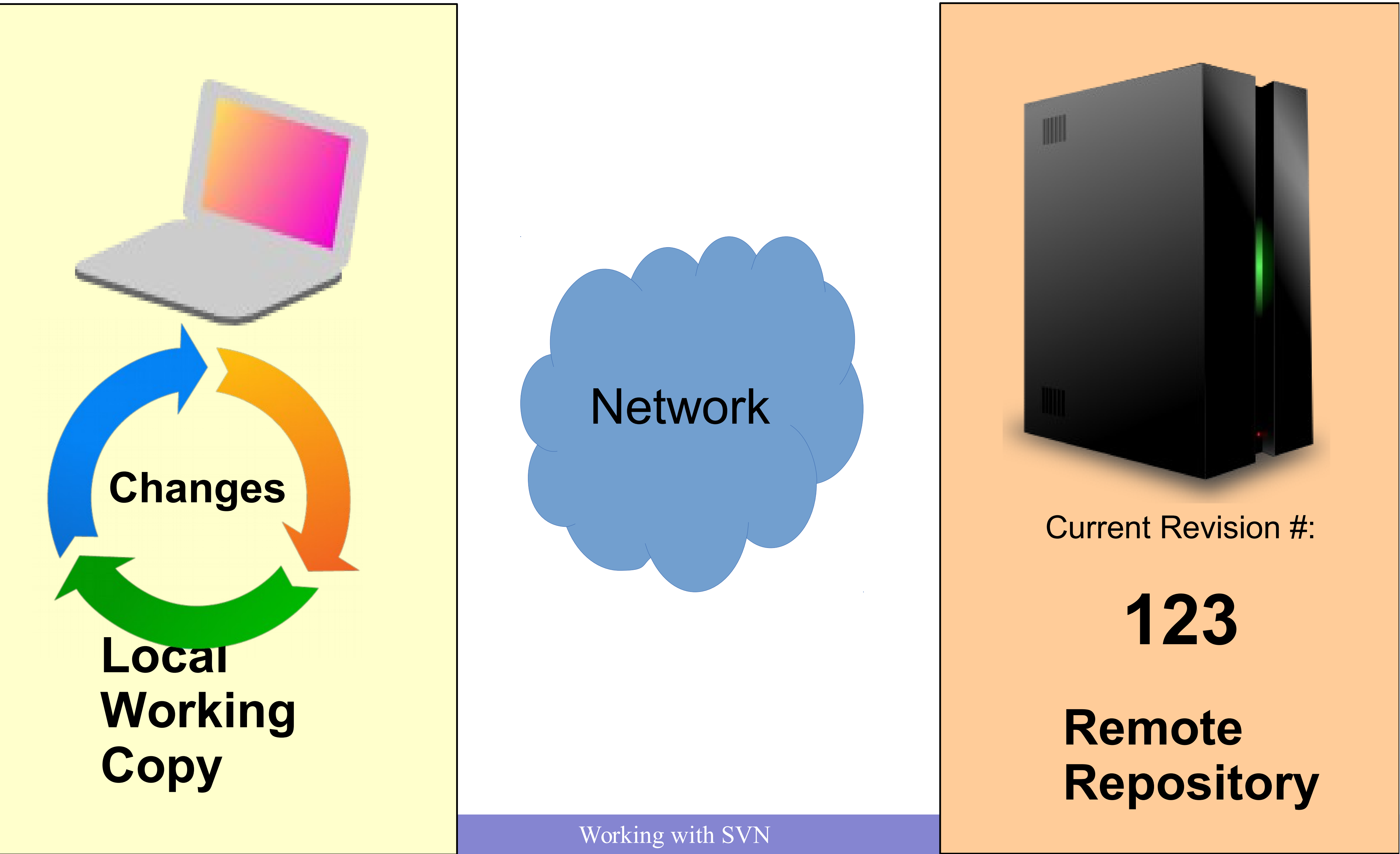
Concepts of version systems



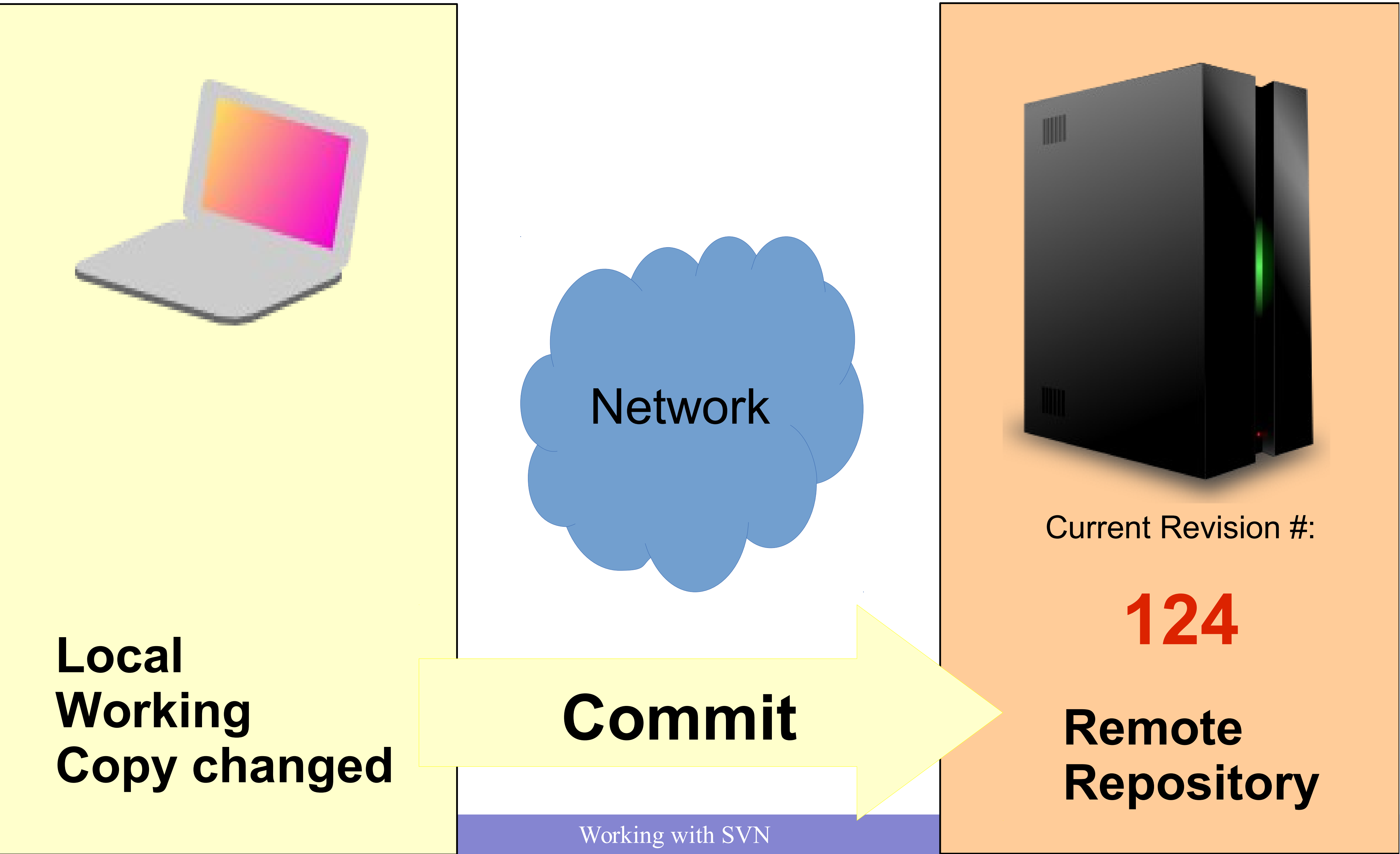
1. Checkout existing code from repo



2. Make changes in the working copy



3. Commit a new version/revision



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 working copies:
  - `mdkir ~/svn/`
  - `cd ~/svn`

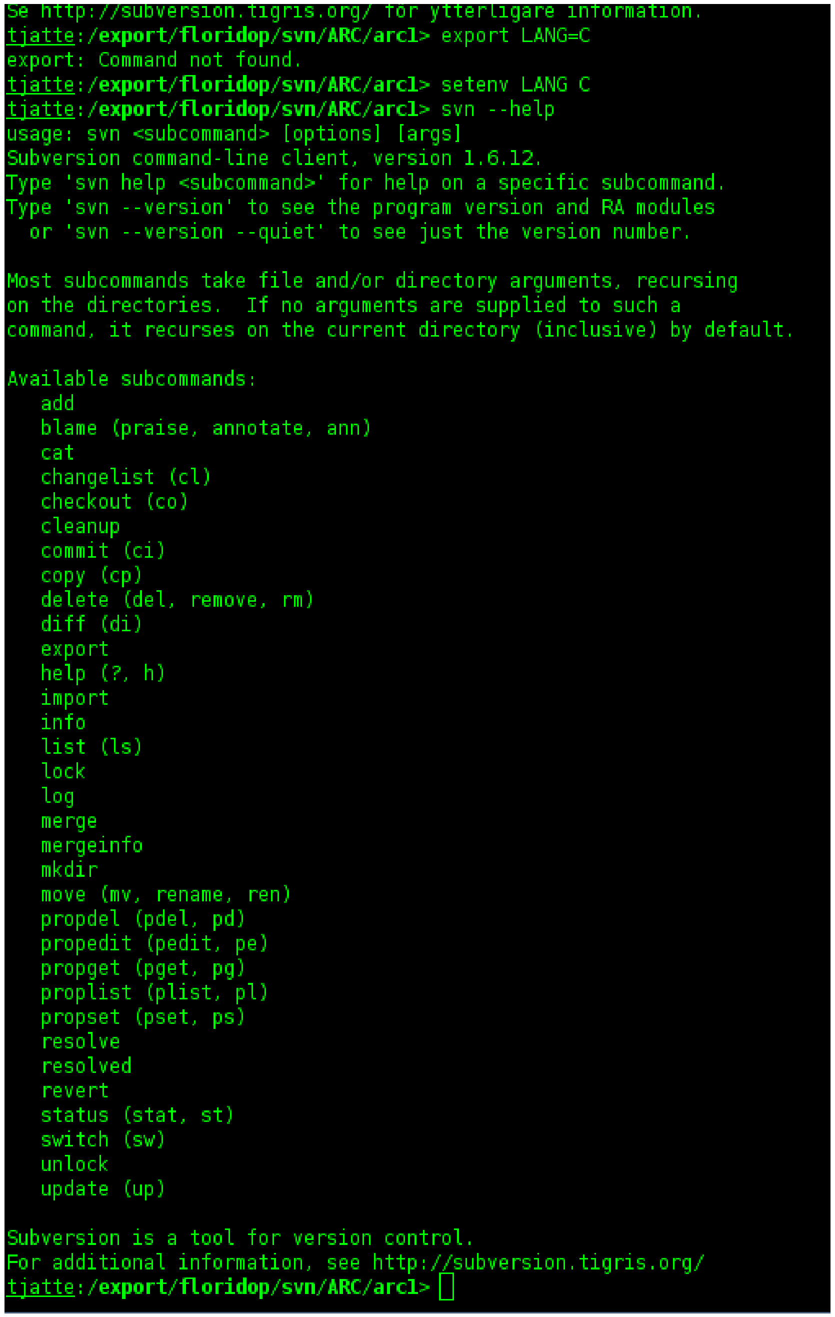
Subversion (SVN)

- 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

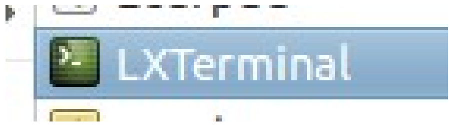
- Checkout from a repository
- Add files to the working copy
- Commit changes to a repository
- Check changes
  - Diffing
  - Reverting
  - Merging
  - Resolution of conflicts
- How to use it for your own code
- Graphical clients
- Homework
- Advanced topics (If spare time)
  - Reverting method 2
  - Creating and applying patches
  - Fork, Branch, Tag

What svn commands are available?



```
Se http://subversion.tigris.org/ for ytterligare information.
tjatte:/export/floridop/svn/ARC/arc1> export LANG=C
export: Command not found.
tjatte:/export/floridop/svn/ARC/arc1> setenv LANG C
tjatte:/export/floridop/svn/ARC/arc1> 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/arc1>
```

Open a terminal.

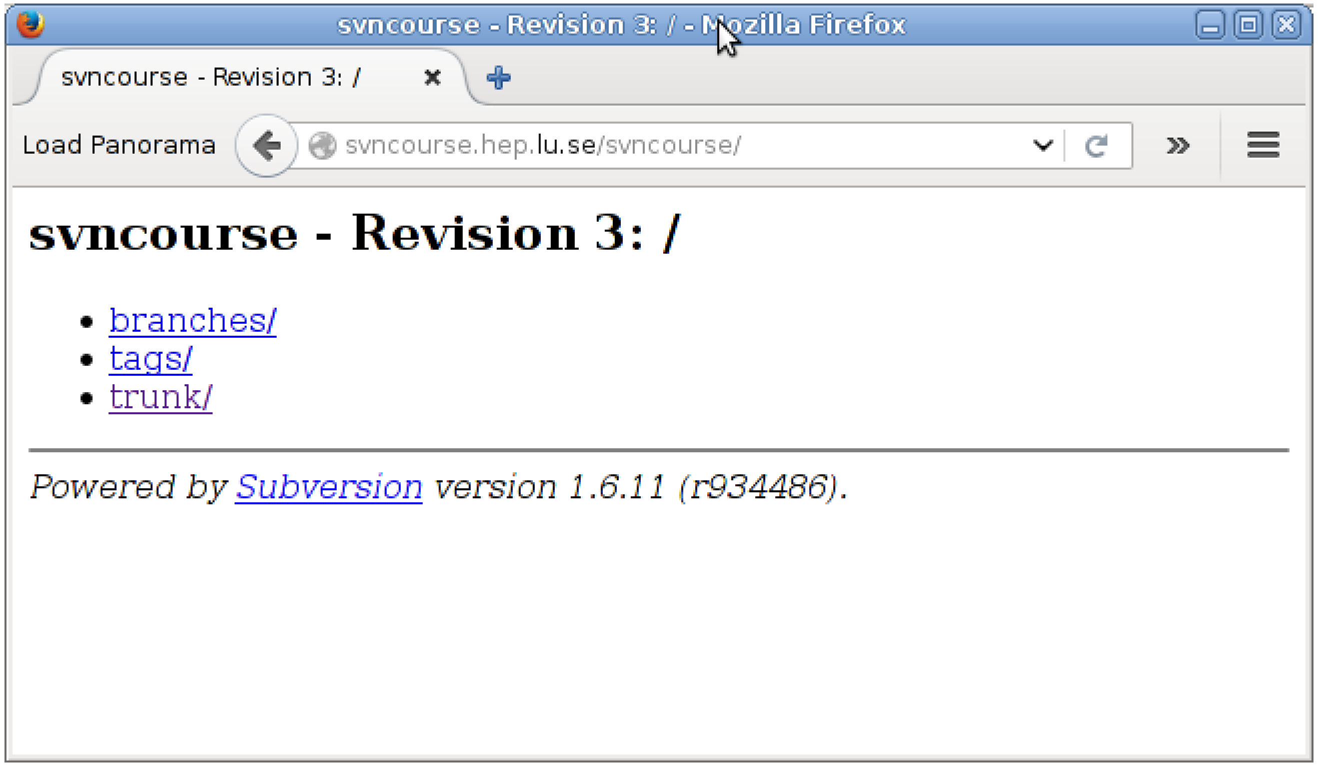


Run the following commands:

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

Explore the content of the course svn server

- Open the browser and go to http://svncourse.hep.lu.se/svncourse/


svncourse - Revision 3: / - Mozilla Firefox
svncourse - Revision 3: /
Load Panorama
svncourse.hep.lu.se/svncourse/

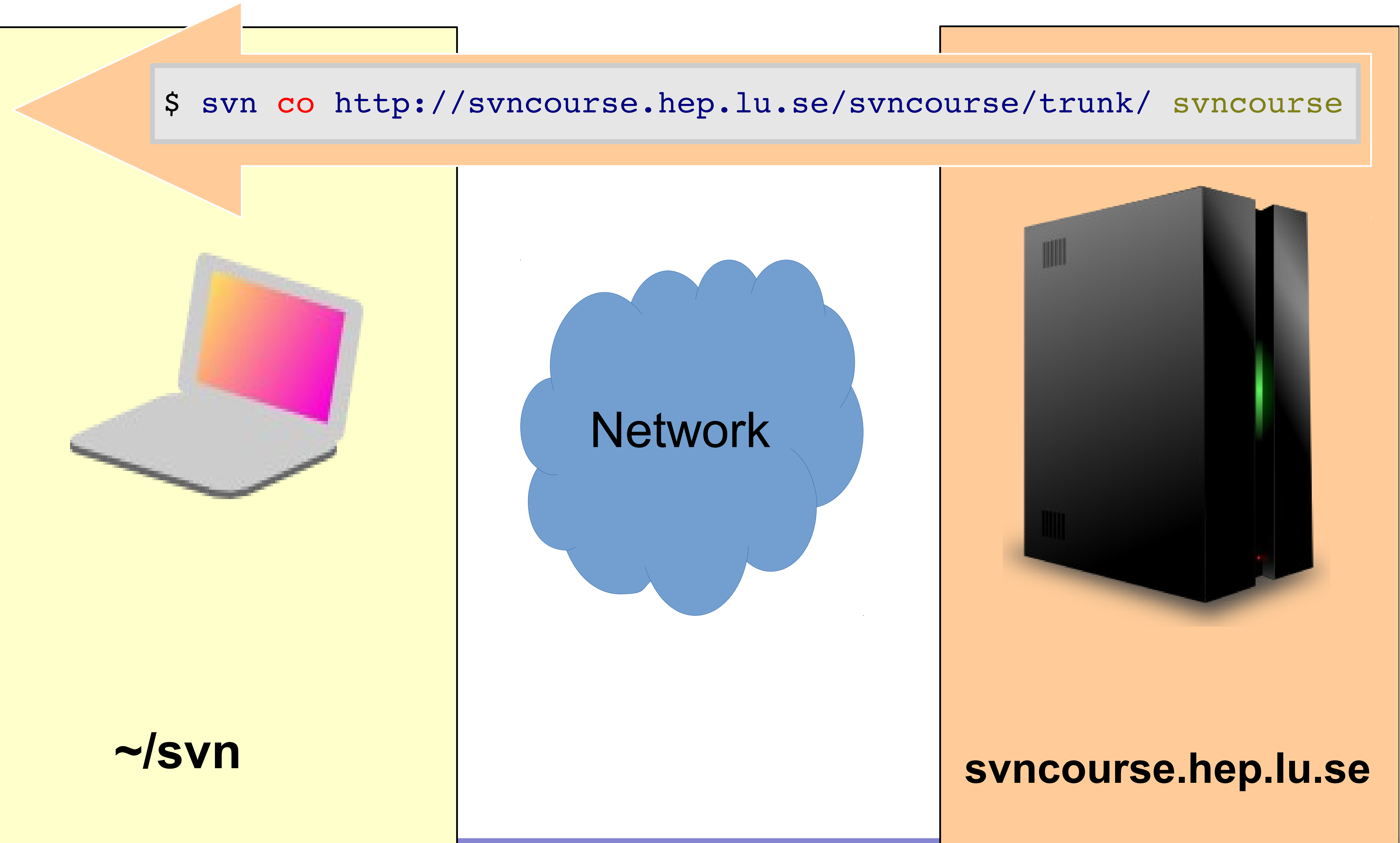
svncourse - Revision 3: /

- branches/
- tags/
- trunk/

Powered by Subversion version 1.6.11 (r934486).

During the lecture you can refresh this page to browse changes

SVN checkout



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

SVN checkout

```
> svn co http://svncourse.hep.lu.se/svncourse/trunk svncoursetrunk
Checked out revision 3.
```

- `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
- `svncoursetrunk`
  The local (on the virtual machine) folder that will be created upon checkout
- **Revision**: a number assigned to a defined version of the code, that gets incremented at every **commit**.

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

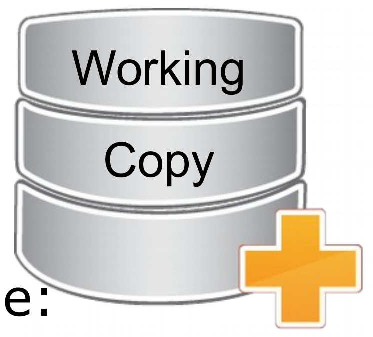
Inspect the working copy

```
> cd svncoursetrunk
> ls -ltrah
total 16K
drwx--x--x 3 courseuser courseuser 4,0K nov  4 16:34 ..
-rw------- 1 courseuser courseuser   45 nov  4 16:37 HELLO.TXT
drwx--x--x 3 courseuser courseuser 4,0K nov  4 16:37 .
drwx--x--x 6 courseuser courseuser 4,0K nov  4 16:37 .svn
```

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

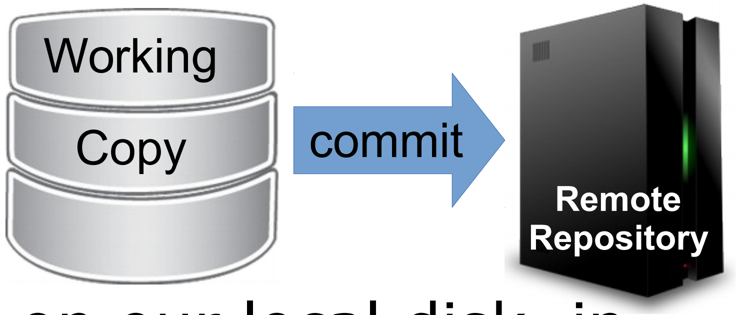
```
> svn info
Path: .
URL: http://svncourse.hep.lu.se/svncourse/trunk
Repository Root: http://svncourse.hep.lu.se/svncourse
Repository UUID: 007b2b91-cb45-42be-b023-e64251eccede
Revision: 2
Node Kind: directory
Schedule: normal
Last Changed Author: balazsk
Last Changed Rev: 2
Last Changed Date: 2015-11-04 16:37:00 +0100 (ons, 04 nov 2015)
```

Ex. 1: Add files



- Inside trunk, create a folder with your username. Example:
  ```
  mkdir floridop
  ```
- Run
  ```
  svn status
  ```
  What does the output mean? Let's discover:
- An svn file can be in different statuses: use
  ```
  svn help status
  ```
  to discover them. What is the status of our folder?
- The file we just created is not yet in the working copy database. We must add it with
  `svn add floridop` (use your folder name here)
- Check `svn status` now. What happens? What does the status value mean? Check again with svn help status.

Ex. 2: Commit



- Up to now, the files are only staying on our local disk, in the working copy. But we want to backup and share them, hence save them back on a remote repository!
- We will also leave a nice message describing what we just committed, using the `-m` option
- Run

```
svn commit --username=floridop -m “my first commit”
```

Using the username I just gave you.
When asked, type the password (case sensitive):

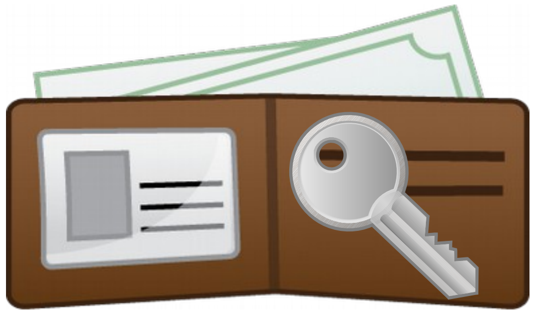
```
svncourse2015
```

```
Password for 'floridop':
```

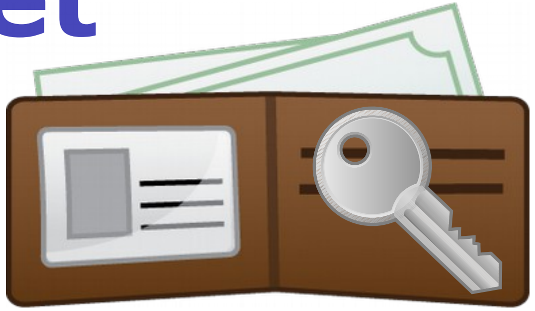
Shortcut: `svn ci`

Intermezzo: a unrelated feature(?): the **password keyring/wallet**

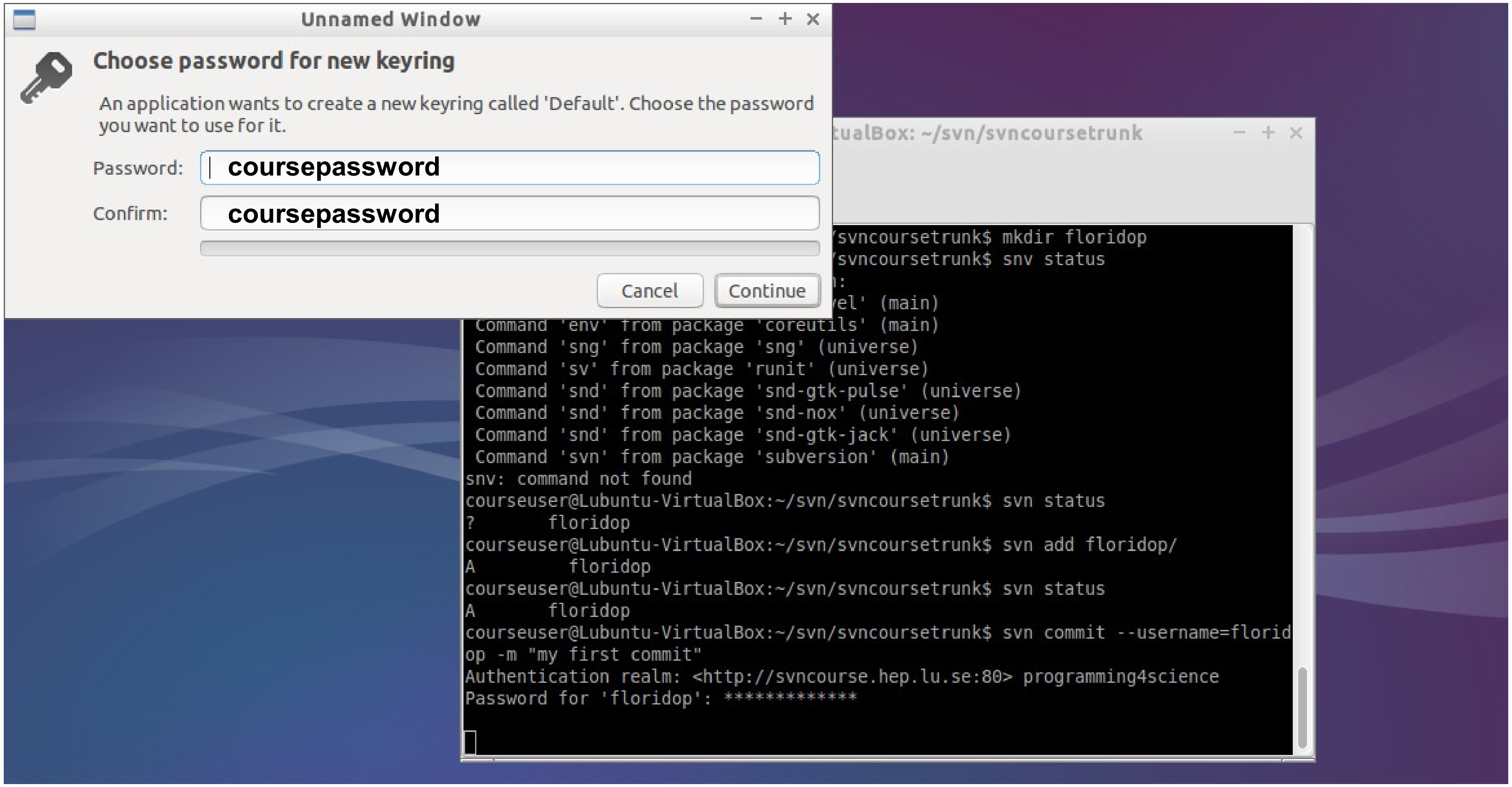
- This has **NOTHING to do with SVN** but is the default behavior on modern distributions
- Stores your password securely, but to enable it you need: a password
- It will insert passwords for you without the need for you to remember them (this is actually dangerous in many ways security-wise… but practical indeed.)
- I suggest you write "coursepassword" when asked. This is only local to the virtual machine, has nothing to do with SVN.



Intermezzo: a unrelated feature(?): the **password wallet**



- Gnome-keyring



Choose password for new keyring

An application wants to create a new keyring called 'Default'. Choose the password you want to use for it.

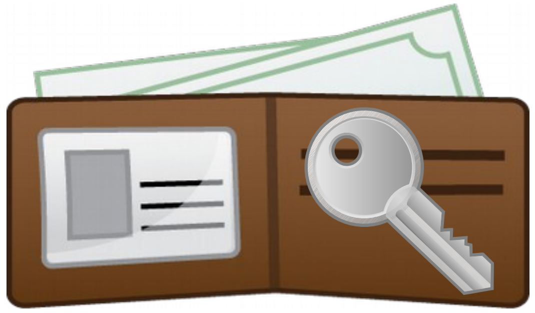
Password: coursepassword

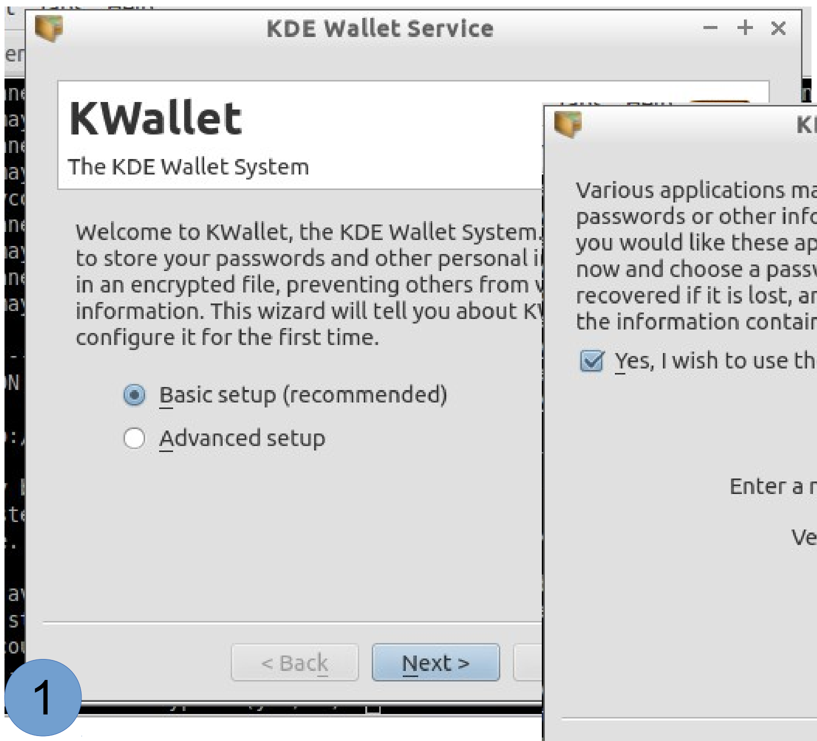
Confirm: coursepassword

```
tualBox: ~/svn/svncoursetrunk
/svncoursetrunk$ mkdir floridop
/svncoursetrunk$ snv status
n:
vel' (main)
 Command 'env' from package 'coreutils' (main)
 Command 'sng' from package 'sng' (universe)
 Command 'sv' from package 'runit' (universe)
 Command 'snd' from package 'snd-gtk-pulse' (universe)
 Command 'snd' from package 'snd-nox' (universe)
 Command 'snd' from package 'snd-gtk-jack' (universe)
 Command 'svn' from package 'subversion' (main)
snv: command not found
courseuser@Lubuntu-VirtualBox:~/svn/svncoursetrunk$ svn status
?       floridop
courseuser@Lubuntu-VirtualBox:~/svn/svncoursetrunk$ svn add floridop/
A         floridop
courseuser@Lubuntu-VirtualBox:~/svn/svncoursetrunk$ svn status
A       floridop
courseuser@Lubuntu-VirtualBox:~/svn/svncoursetrunk$ svn commit --username=florid
op -m "my first commit"
Authentication realm: <http://svncourse.hep.lu.se:80> programming4science
Password for 'floridop': *************
```

Intermezzo: a unrelated feature(?): the **password wallet**

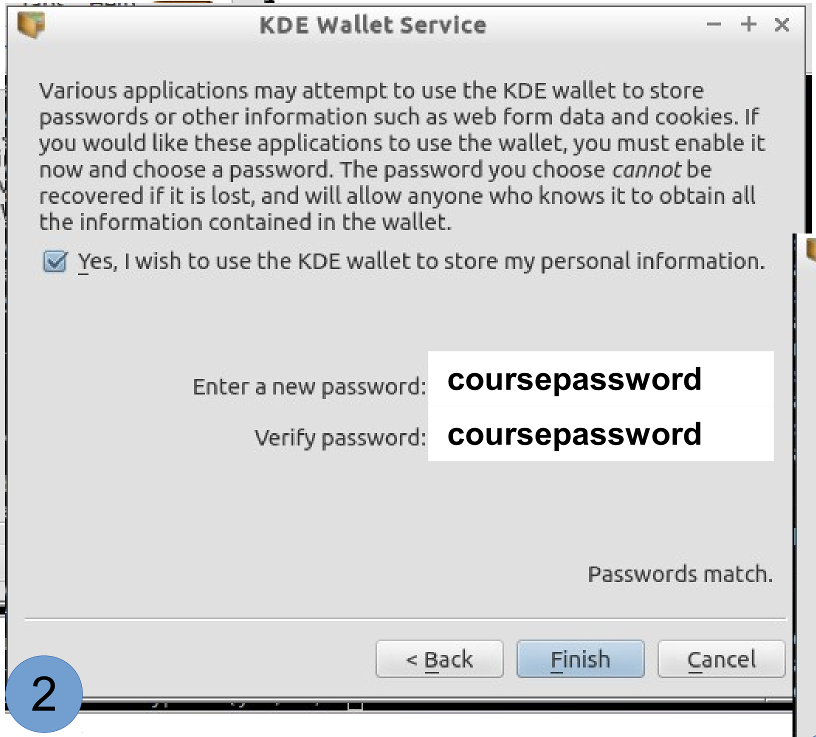
- Kde-wallet





Welcome to KWallet, the KDE Wallet System to store your passwords and other personal i in an encrypted file, preventing others from v information. This wizard will tell you about K configure it for the first time.

(● Basic setup (recommended)) ○ Advanced setup



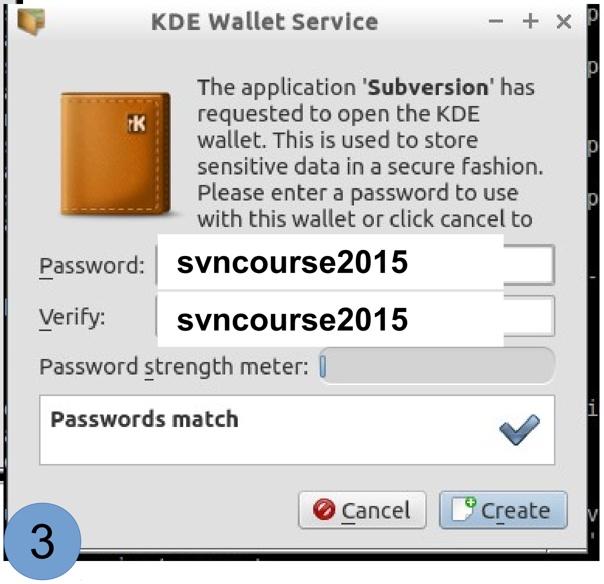
Various applications may attempt to use the KDE wallet to store passwords or other information such as web form data and cookies. If you would like these applications to use the wallet, you must enable it now and choose a password. The password you choose *cannot* be recovered if it is lost, and will allow anyone who knows it to obtain all the information contained in the wallet.

✓ Yes, I wish to use the KDE wallet to store my personal information.

Enter a new password: **coursepassword**

Verify password: **coursepassword**

Passwords match.



The application '**Subversion**' has requested to open the KDE wallet. This is used to store sensitive data in a secure fashion. Please enter a password to use with this wallet or click cancel to

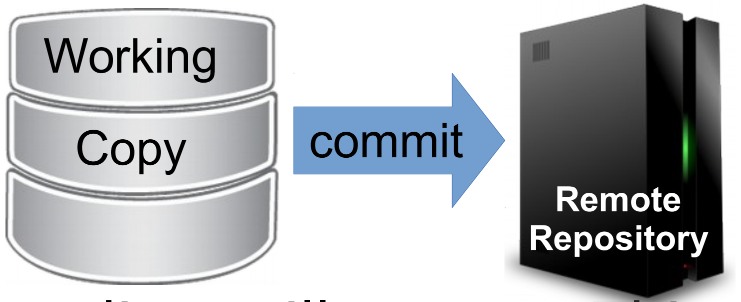
Password: **svncourse2015**

Verify: **svncourse2015**

Password strength meter:

Passwords match ✓

Ex. 2: Commit



- If you don't specify the `-m` option, 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.

• OPTIONAL: the file editor can be changed. I prefer to use `-m` on the command line. But if you want to use an editor, like geany: For example, to use geany, execute:

```
export SVN_EDITOR=geany
```

```
Password for 'floridop':
Adding          floridop
Committed revision 3.
```

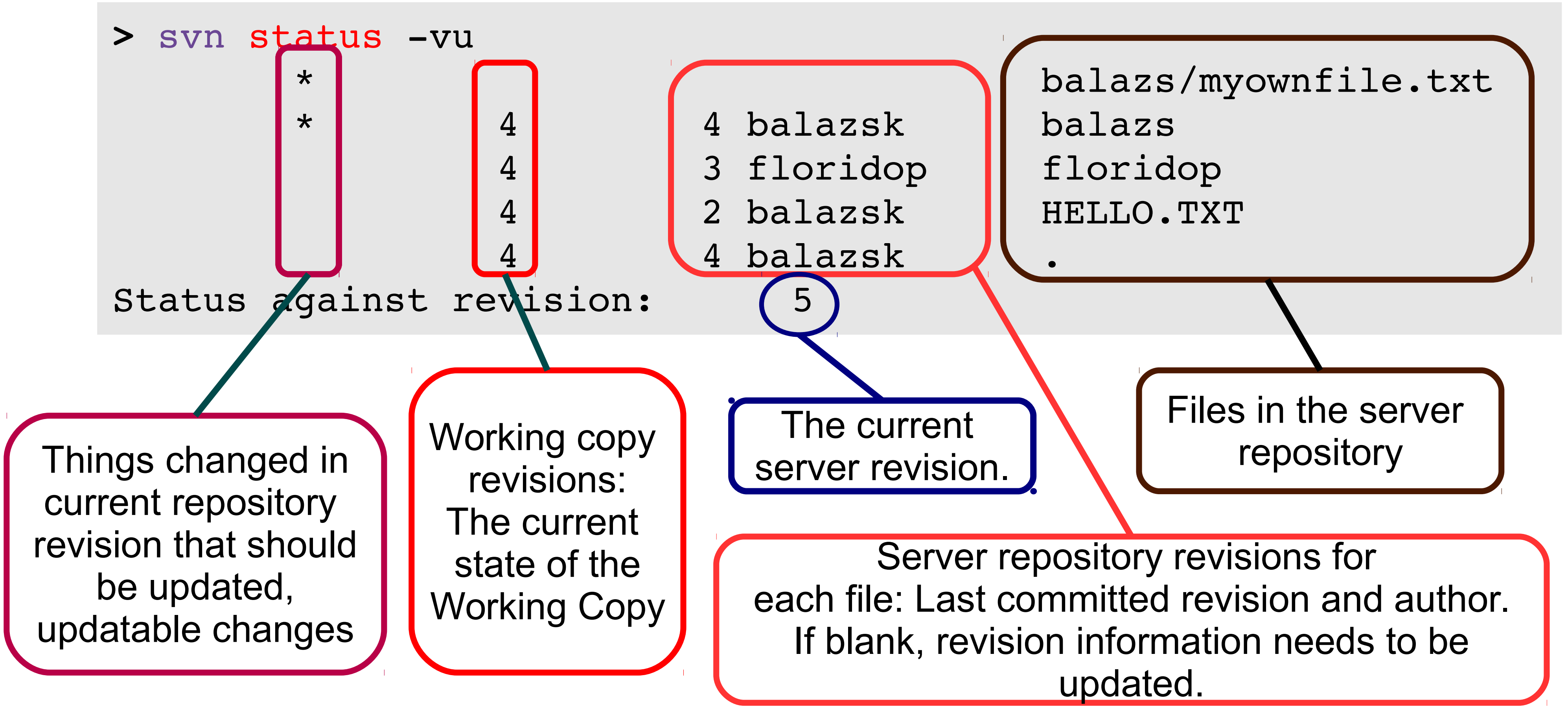
Commit – what happened?

- Run

  `svn status -vu`

Shortcut: `svn st -vu`

- It shows the updates pending in the repository and other info:

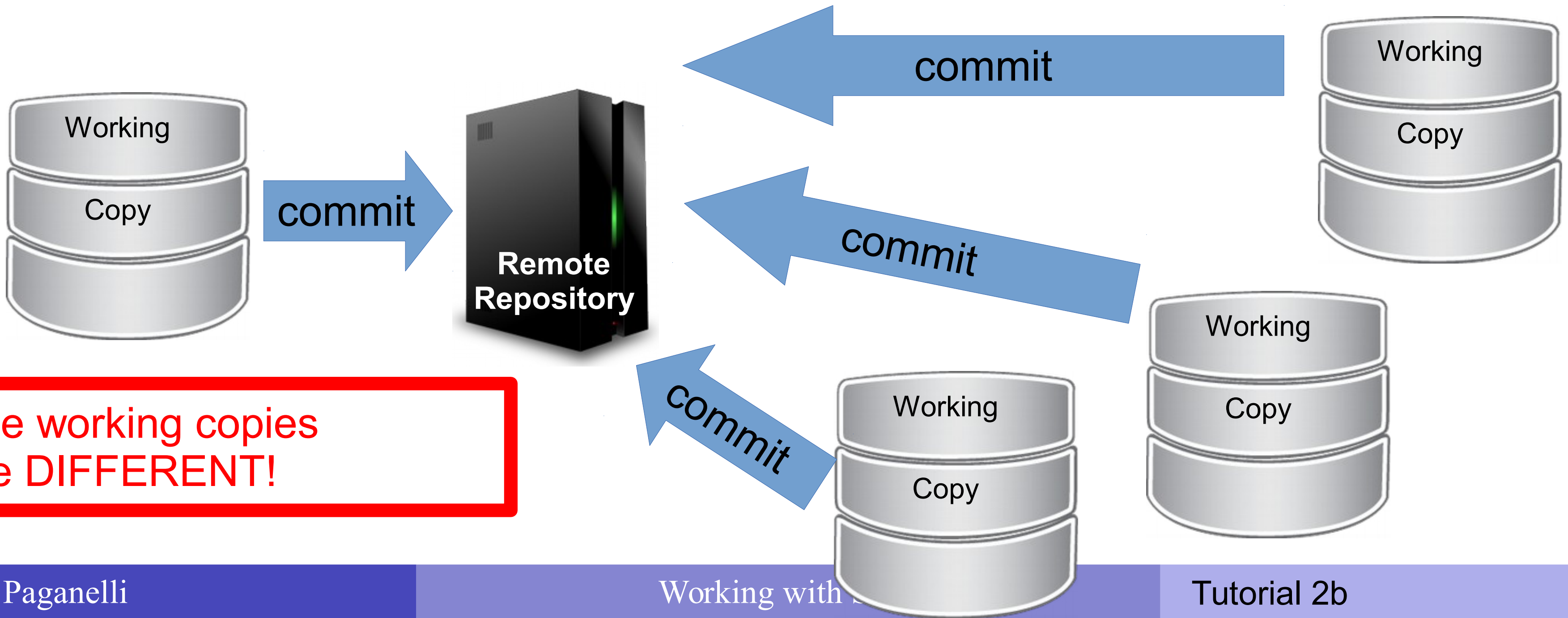


```
> svn status -vu
        *                                       balazs/myownfile.txt
        *        4        4 balazsk             balazs
                 4        3 floridop            floridop
                 4        2 balazsk             HELLO.TXT
                 4        4 balazsk             .
Status against revision:      5
```

Things changed in current repository revision that should be updated, updatable changes
Working copy revisions: The current state of the Working Copy
The current server revision.
Files in the server repository
Server repository revisions for each file: Last committed revision and author. If blank, revision information needs to be updated.

Commit – what happened?

- Check workspace information - run
  `svn info`
- Check server information - run
  `svn info http://svncourse.hep.lu.se/svncourse/trunk`
- Discuss the differences with the teacher.

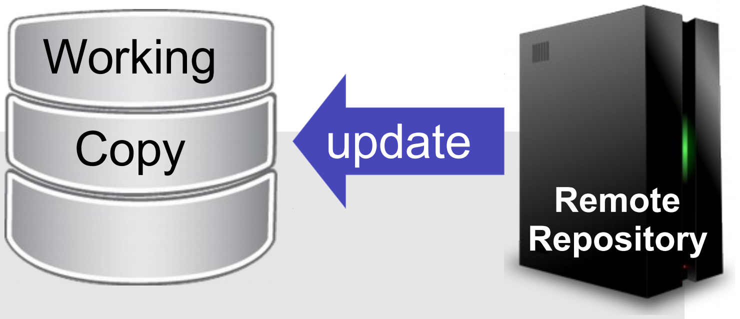


The working copies are DIFFERENT!

Ex. 3: Synch with the server: update

- We need to update the content of our work area with the actual status of the central server. This is done with:

  svn update



```
$> svn update
A     balazs
A     balazs/myownfile.txt
A     floridop
Updated to revision 5.

> svn status -uv
                 5         2 balazsk      HELLO.TXT
                 5         5 balazsk      balazs/myownfile.txt
                 5         5 balazsk      balazs
                 5         3 floridop     floridop
                 5         5 balazsk      .
Status against revision:      5
```

Shortcut: **svn up**

Ex. 4: The commit log

- Keeps track of the commits
- Run

  ```
  svn log -v
  ```

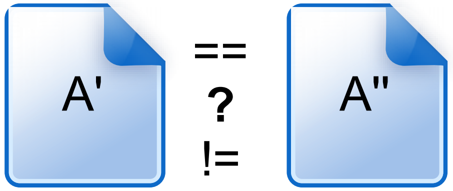
  to see it

```
> svn log -v
------------------------------------------------------------------------
r5 | balazsk | 2015-11-06 17:49:01 +0100 (fre, 06 nov 2015) | 2 lines
Changed paths:
   A /trunk/balazs/myownfile.txt

Hello the editor stuff did not work...

------------------------------------------------------------------------
r4 | balazsk | 2015-11-06 17:34:13 +0100 (fre, 06 nov 2015) | 1 line
Changed paths:
   A /trunk/balazs

my first commit
------------------------------------------------------------------------
r3 | floridop | 2015-11-06 17:28:32 +0100 (fre, 06 nov 2015) | 1 line
Changed paths:
   A /trunk/floridop
...
```

Ex. 5: create a file and commit

- **Best practice: update first, and then commit!**
  1. **before** changing anything, always do an **`update`**, so that you're sure you're working on the latest version of a file.
  2. Then you're safe to **`commit`**.
- Exercise:
  1. Update (`svn update`)
  2. `cd` into the folder with your name and create a file.
  3. Add the file to the versioning system (`svn add ...`)
  4. run `svn status -uv` and compare revisions
  5. Commit (`svn commit –username=... -m “write a description”`)
  6. run `svn status -uv` again and discuss with the teacher.

Ex. 6: Diffing



- Make some change in the file in your working copy.
- Check `svn status -uv`
- Run

  `svn diff`

```
> svn diff
Index: thisisfloridofile.txt
===================================================================
--- thisisfloridofile.txt (revision 6)
+++ thisisfloridofile.txt (working copy)
@@ -1 +1,2 @@
 Hello! this is florido's file.
+I am adding this change.
```

Line numbers

Shortcut: **`svn df`**

Ex. 7: 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 thisisfloridofile.txt
svn diff
```

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

Ex. 8: 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 – one is in the advanced topics at the end of these slides.

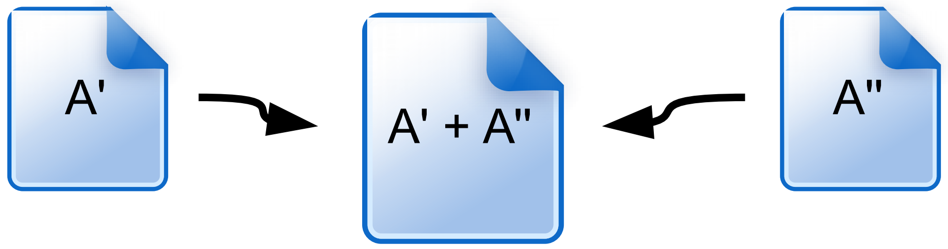
Ex. 8: Reverting to a previous revision method 1: export

- 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 – that is, **overwriting another revision of the file on top of the current**.
- **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 thisisfloridofile.txt .
    ```

    will roll back `thisisfloridofile.txt` to revision 3 in the folder `.` (current folder)
  - **`svn diff`**
  - **`svn commit`** the changes

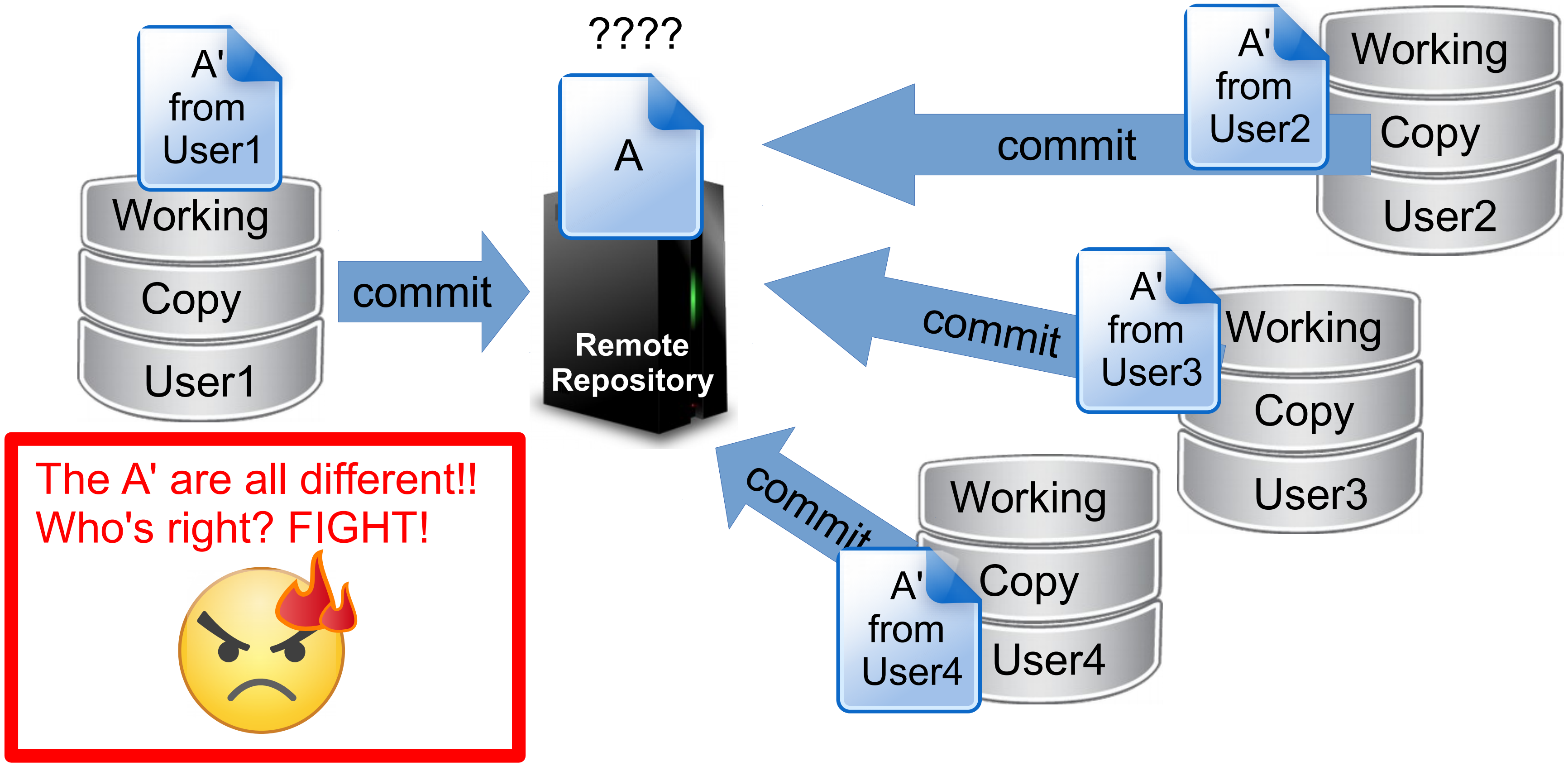
Merging



- 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 mixed 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!

Conflicts

- A conflict happens when somebody edits a file that somebody else edited starting from the same version and committed at the same time,
  or tries to commit without UPDATING first.



- 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.

Ex. 9: Let's generate a conflict!

- Open and add some text to `conflictfile.txt` that I just created. (run `svn update` to get it!)
- It should contain:
  - Your name
  - A sentence of your choice
  - Make it just one line please!
- 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.

Typical conflict commit error

- If you see this, very likely the file you just edited has been modified and updated to a new revision.

```
> svn ci --username=floridop -m "this is florido's line"
Sending        conflictfile.txt
svn: E160024: Commit failed (details follow):
svn: E160024: File 'conflictfile.txt' is out of date; try updating
svn: E160024: resource out of date; try updating
```

- The solution is to adhere to the SVN Golden Rule: ALWAYS **UPDATE** FIRST, THEN **COMMIT**!

Ex. 10: Conflicts resolution

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

```
> svn up
Conflict discovered in 'conflictfile.txt'.
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
```

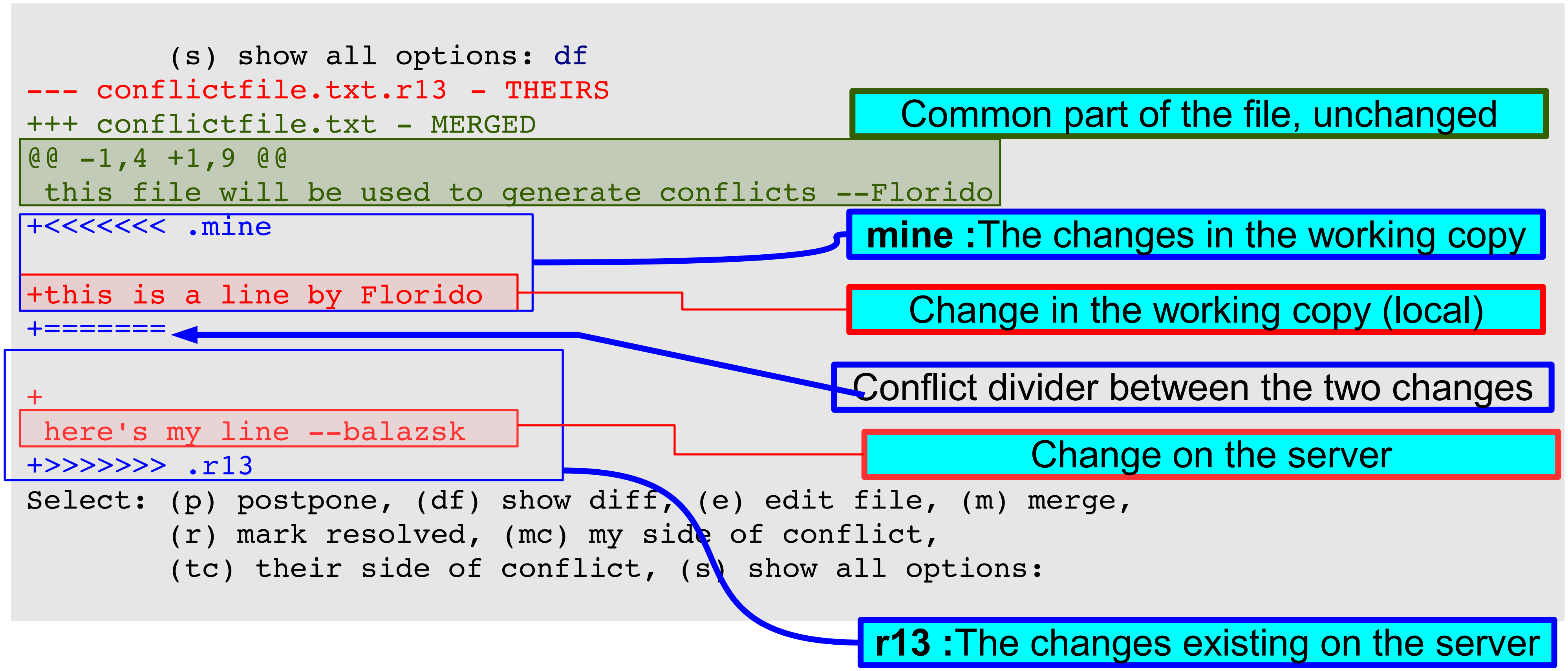
Ex. 10: Conflicts resolution - diff

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

```
        (s) show all options: df
--- conflictfile.txt.r13 - THEIRS
+++ conflictfile.txt - MERGED
@@ -1,4 +1,9 @@
 this file will be used to generate conflicts --Florido
+<<<<<<< .mine

+this is a line by Florido
+=======

+
 here's my line --balazsk
+>>>>>>> .r13
Select: (p) postpone, (df) show diff, (e) edit file, (m) merge,
        (r) mark resolved, (mc) my side of conflict,
        (tc) their side of conflict, (s) show all options:
```

Ex. 10: Conflicts resolution - diff

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



```
        (s) show all options: df
--- conflictfile.txt.r13 - THEIRS
+++ conflictfile.txt - MERGED
@@ -1,4 +1,9 @@
 this file will be used to generate conflicts --Florido
+<<<<<<< .mine

+this is a line by Florido
+=======

+
 here's my line --balazsk
+>>>>>>> .r13
Select: (p) postpone, (df) show diff, (e) edit file, (m) merge,
        (r) mark resolved, (mc) my side of conflict,
        (tc) their side of conflict, (s) show all options:
```

Common part of the file, unchanged
mine : The changes in the working copy
Change in the working copy (local)
Conflict divider between the two changes
Change on the server
r13 : The changes existing on the server

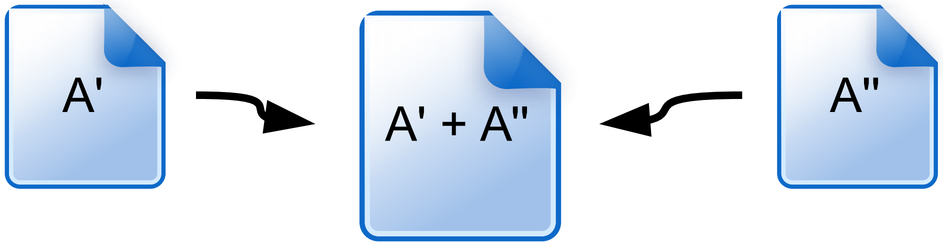
Ex. 10: Conflicts resolution - diff

- **mine-conflict**: select my changes and resolve the 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!

Ex. 10: 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, (p) option**: we will resolve the conflict using `meld`
- List (bash `ls`) the contents of the SVN directory. What happened?

Ex. 11: 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:

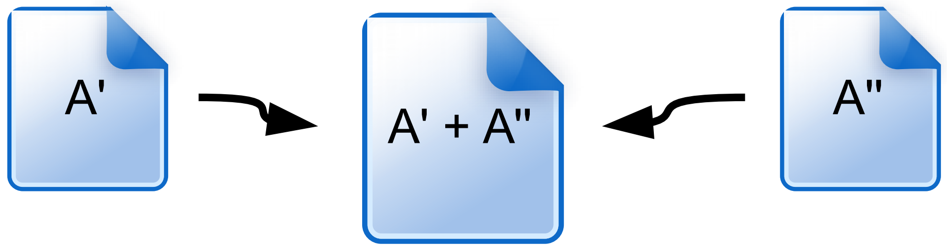
  ```
  meld source1 destination source2
  ```

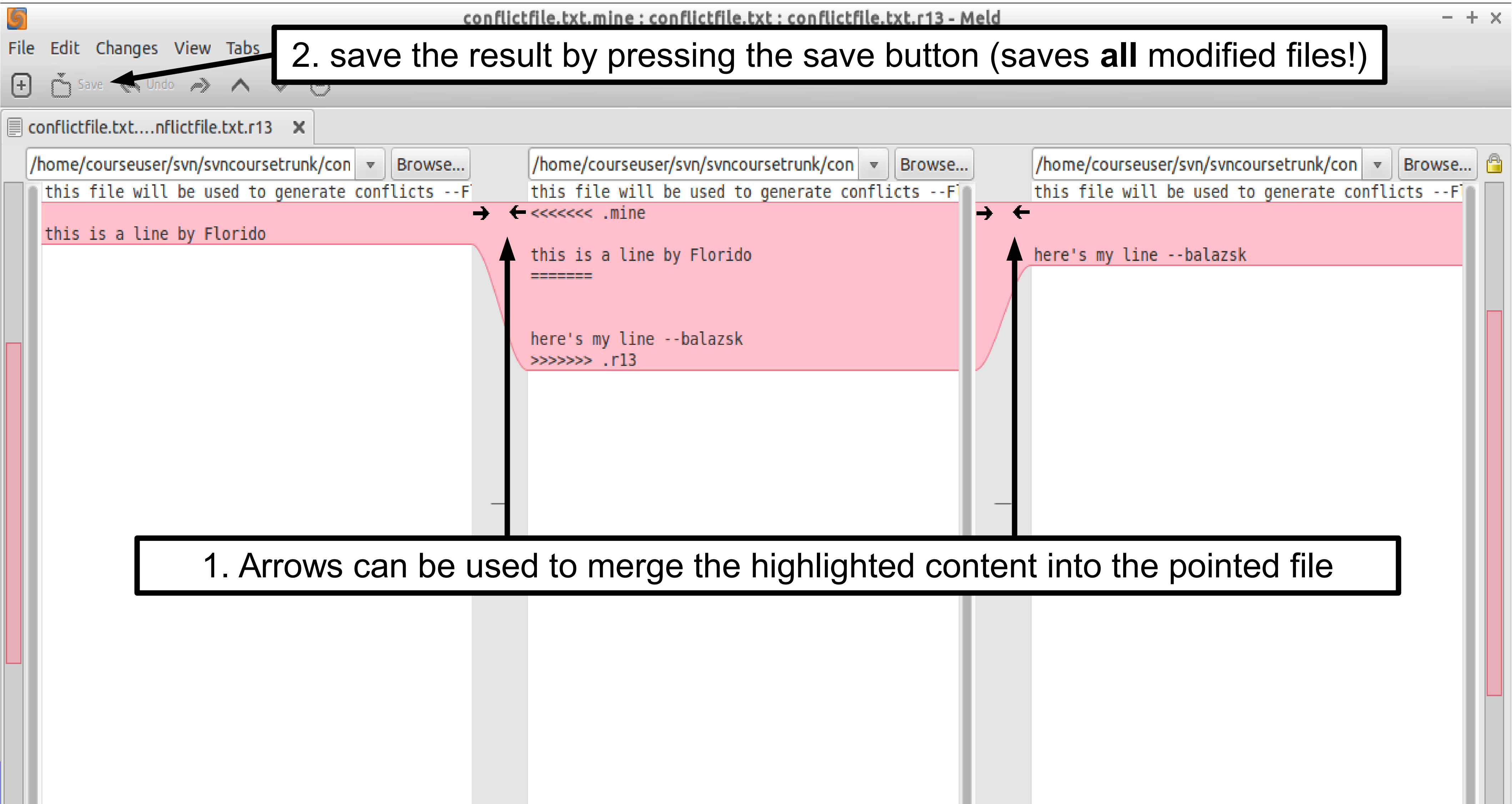
- That means, we want to merge the contents of the files `source1` and `source2` into **`destination`**.
- In our case:

  ```
  meld conflictsfile.txt.mine conflictsfile.txt conflictsfile.txt.r16
  ```

  where `r16` is revision number that conflicts, written by SVN when we chose **postpone**.
- Run it!

Merging with `meld`





conflictfile.txt.mine : conflictfile.txt : conflictfile.txt.r13 - Meld
File
Edit
Changes
View
Tabs
Save
Undo
conflictfile.txt....nflictfile.txt.r13
Browse...

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

```
/home/courseuser/svn/svncoursetrunk/con
this file will be used to generate conflicts --F

this is a line by Florido
```

```
/home/courseuser/svn/svncoursetrunk/con
this file will be used to generate conflicts --F
<<<<<<< .mine

this is a line by Florido
=======


here's my line --balazsk
>>>>>>> .r13
```

```
/home/courseuser/svn/svncoursetrunk/con
this file will be used to generate conflicts --F


here's my line --balazsk
```

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

EX 12: 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 conflictfile.txt`

- After this, we're ready to commit.

```
$ svn commit --username=floridop -m "resolved conflict by adding one line per user"
svn: E155015: Commit failed (details follow):
svn: E155015: Aborting commit: '/home/courseuser/svn/svncoursetrunk/conflictfile.txt'
              remains in conflict

$ svn resolved conflictfile.txt
Resolved conflicted state of 'conflictfile.txt'
$ svn commit --username=floridop -m "resolved conflict by adding one line per user"
Sending        conflictfile.txt
Transmitting file data .
Committed revision 14.
```

Ex 13: 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 "Intial import of files"
```

- From now on you can checkout the repository using

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

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

Ex. 14: Graphical Clients

- Want to try a graphical client?
  - Minimalistic one: run

    ```
    rapidsvn
    ```

    - This one is available in Lubuntu repositories. Install line: `sudo apt-get install rapidsvn`
  - Feature-rich one (not available in repositories):

    ```
    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 if you want the latest version.
      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

Importance of SVN within the course

- **Problem**: the virtual machine disk you're using can be wiped all time, and there is no guarantee the files you left there will be kept.
- **Solution**: From this tutorial on, you're invited to put your code files on the SVN server at the end of each tutorial session.
  - Suggestion: create a directory `TutorialXY` in your /username/ SVN folder for each tutorial
- We promise to keep your files on the SVN server for the duration of the course and course project.
- The final course project material you will create can be only handed out using a special SVN server we will indicate, so it is good to get aquainted with SVN during the course.

Homework Tutorial 2b

- Install and configure one of SVN graphical clients. It does not necessarily have to be any of those mentioned in this tutorial.
- Checkout your work folder from the trunk of svncourse. i.e. from the URL
http://svncourse.hep.lu.se/svncourse/trunk/username/
- Commit at least one LaTeX file you created during Tutorial 2a.
  - ! Describe which client you used in the commit log.

References

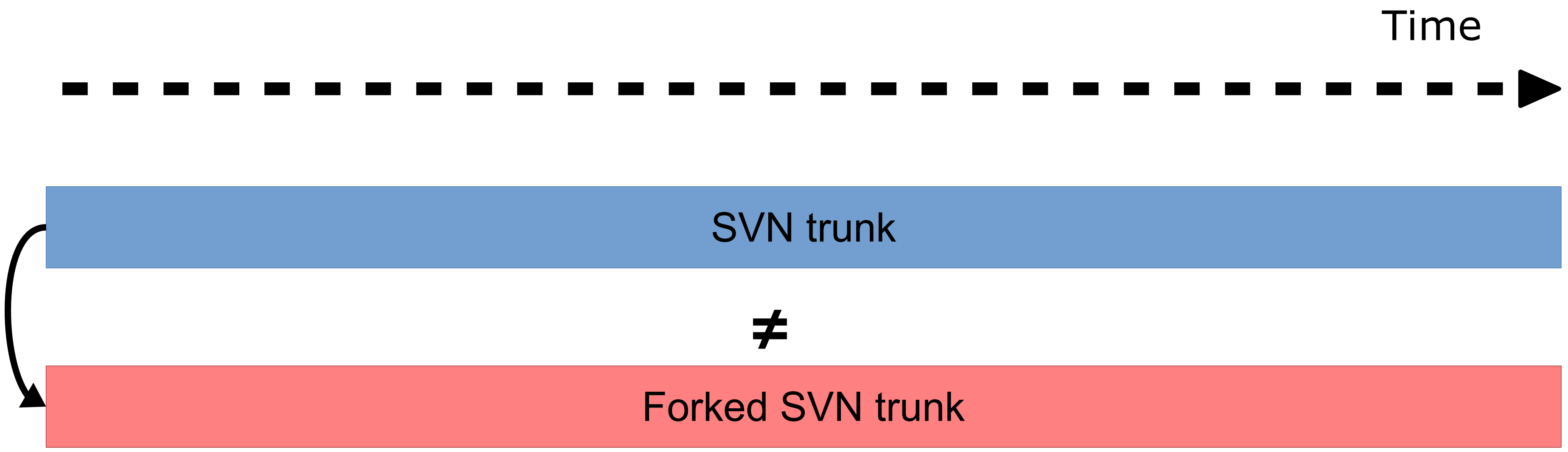
- SVN Quick Reference Card:
  http://wiki.ssg.uab.edu/download/attachments/3080576/Subversion+Quick+Reference+Card.pdf?version=1
- The SVN Redbook
  http://svnbook.red-bean.com/
- Patching with SVN:
  https://ariejan.net/2007/07/03/how-to-create-and-apply-a-patch-with-subversion/

Pictures references

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

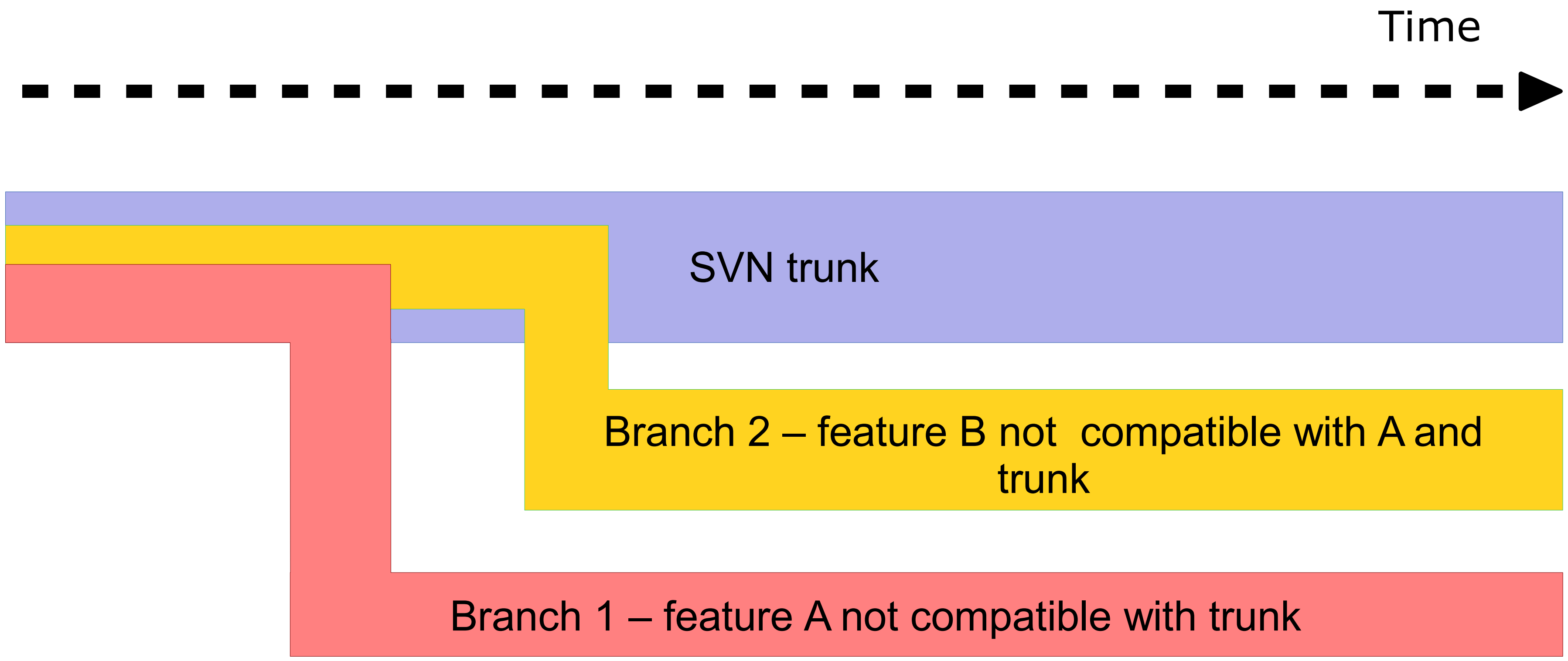
Advanced topics

Fork



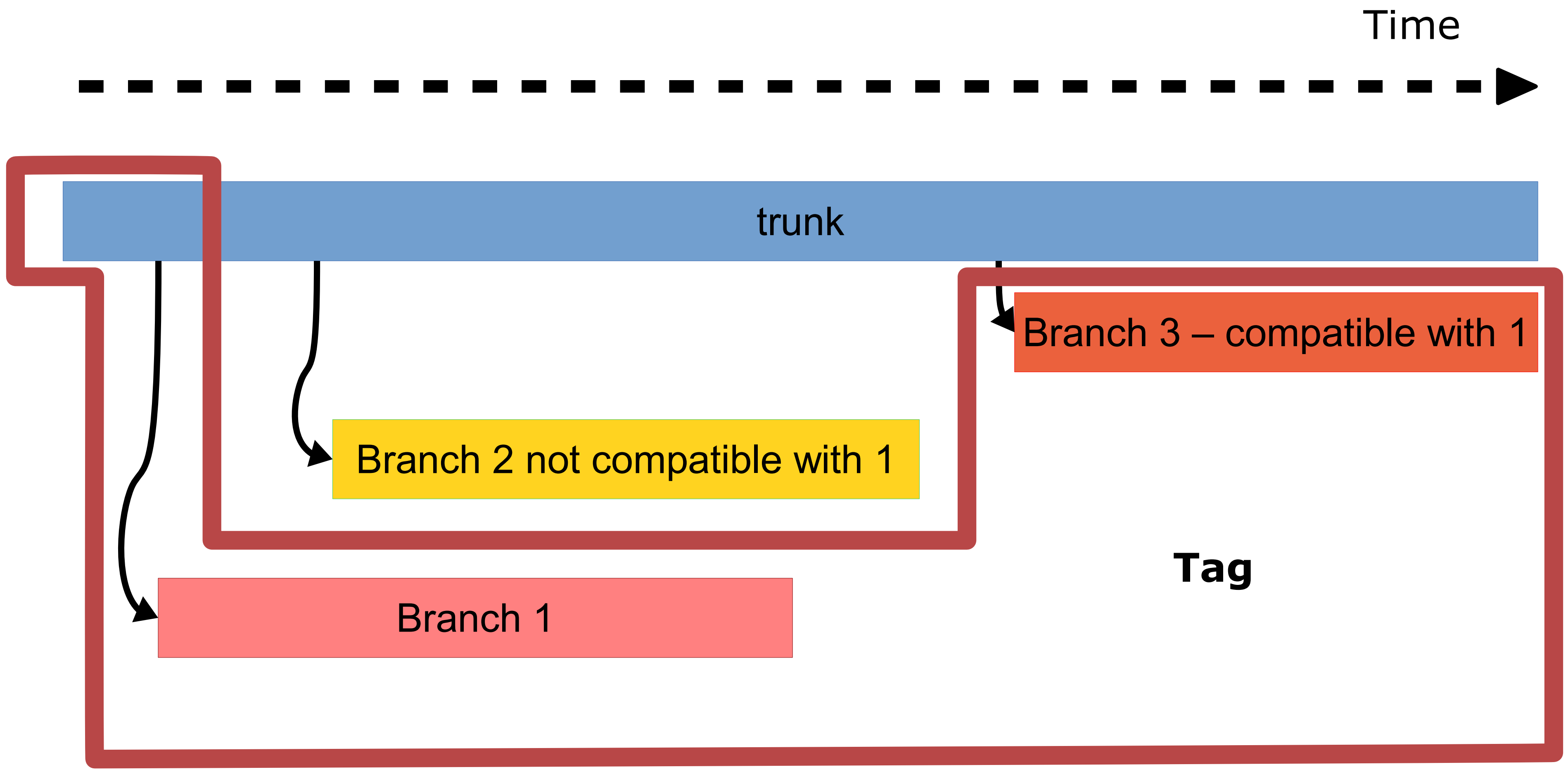
To copy a whole trunk into another working copy, to create a completely different program from the existing one.

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.

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 thisisfloridofile.txt
```

- 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`.
- Commit the changes to update the server database.

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 `conflictfile.txt` to a friend.
- Later in time, we change its contents.
- We would like to give the new version to a friend, but he/she/ze:
  - Does not want to use SVN
  - He/she/ze has very limited space to carry the new code around, for example on a usb pen. We would just like to share the newer parts, what changed.

Creating patches wit svn diff

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

```
   svn diff -r conflictfile.txt@8
conflictfile.txt@HEAD
```

- This generates a patch file output. What we have to do is write the output to a file (see lecture about the shell!):

```
svn diff -r conflictfile.txt@8
conflictfile.txt@HEAD >
conflictfile.txt.20151112.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 8 of `conflictfile.txt` to test the patch.
  - Create a folder in your home
    ```
    mkdir ~/test/
    ```
  - Export to that folder `conflictfile.txt` at revision 3 with `svn export` (check previous slides!)
  - copy the **`conflictfile.txt.20151112.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 **`conflictfile.txt.20151112.patch`** files are in the `~/test/` folder.
- `cat` the content of `conflictfile.txt`
- Run the following:

```
patch -p0 -i conflictfile.txt.20151112.patch
```

  - **`-p0`**: go up of 0 directories (it does `cd ../` as many times as the indicated number)
  - **`-i conflictfile.txt.20151112.patch`**: use **`conflictfile.txt.20151112.patch`** as input file that contains instructions how to patch.
- `cat` the content of `conflictfile.txt` again. It changed!