**SVN basic commands to Git commands**

Before going through the SVN to Git commands, please take time to read some basic information about Git. The page "[Git practices, workflows and tips](http://aliceinfo.cern.ch/Offline/node/2972" \t "_blank)" explains how to start working with Git, best practices and workflows.

For any issue with AliRoot Git repository please submit a ticket to [Git administration JIRA project](https://alice.its.cern.ch/jira/browse/GA).

1. [Migrating from SVN to Git](http://aliceinfo.cern.ch/Offline/node/2912/#Migration)
2. [CERN Central Git Service](http://aliceinfo.cern.ch/Offline/node/2912/#CERNGit)
   1. [General description](http://aliceinfo.cern.ch/Offline/node/2912/#cern-general-description)
   2. [CERN Git Service FAQ](http://aliceinfo.cern.ch/Offline/node/2912/#cern-git-faq)
   3. [AliRoot access rules](http://aliceinfo.cern.ch/Offline/node/2912/#aliroot-access)
   4. [AliRoot git web interface](http://aliceinfo.cern.ch/Offline/node/2912/#gitweb)
3. [First-time Git Setup](http://aliceinfo.cern.ch/Offline/node/2912/#firsttime)
4. [SVN to Git commands](http://aliceinfo.cern.ch/Offline/node/2912/#Commands)
   1. [Browse commands](http://aliceinfo.cern.ch/Offline/node/2912/#BrowseCommands)
   2. [Update commands](http://aliceinfo.cern.ch/Offline/node/2912/#UpdateCommands)
   3. [Advanced commands](http://aliceinfo.cern.ch/Offline/node/2912/#AdvancedCommands)
   4. [Storing passwords](http://aliceinfo.cern.ch/Offline/node/2912/#passwords)
5. [Git tips and best practices](http://aliceinfo.cern.ch/Offline/node/2912/#GitTips)
6. [Checking out more than one branch/tag](http://aliceinfo.cern.ch/Offline/node/2912/#ParallelGit)
7. [Useful links](http://aliceinfo.cern.ch/Offline/node/2912/#UsefulLinks)

**1. Migration from SVN to Git**

NOTE: Version Control System = VCS

For a smooth transition from SVN to Git the developers need first to understand the difference between a [centralized VCS](http://blogs.atlassian.com/2012/02/version-control-centralized-dvcs/), such as SVN and a [distributed VCS](https://en.wikipedia.org/wiki/Distributed_revision_control), such as Git.

A centralized VCS uses a central repository that users can copy locally and to which developers commit changes one by one. All users see the same version of the central repository, therefore everyone knows to a certain degree what everyone else on the project is doing. Administrators have fine-grained control over who can do what.

A distributed VCS uses a central repository, containing the official, up-to-date release code, but clients don’t just check out the latest snapshot of the files: they fully mirror the repository. Thus if the central repository dies, any of the client repositories can be copied back up to the server to restore it. Every checkout is really a full backup of all the data. [[1](http://git-scm.com/book/en/Getting-Started-About-Version-Control)].

One of the major advantage using a distributed VCS is that operations like commit, merge or switch are really fast, being done locally. Also due to its archiving algorithms the repository size is smaller than in SVN [[2](https://indico.cern.ch/getFile.py/access?contribId=0&resId=0&materialId=slides&confId=246803)].

One of the biggest disadvantage of a distributed VCS is that you are not able to download only parts of the repository. Being distributed, the user has to download the full repository (clone).

**2. CERN Central Git Service**

**2.1 General description**

CERN IT Department decided to host and support one flavor of [centralized VCS (SVN)](http://svn.web.cern.ch/svn/) and one flavor of [distributed VCS (git)](http://information-technology.web.cern.ch/services/git-service).

Besides the hosting of the service, CERN IT offers also integration with CERN Authentication.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Levels | | | Authentication protocol | URL |
| R | C | W |
| \* | \* | \* | Kerberos1 | [https://git.cern.ch/kerberos/AliRoot](https://:@git.cern.ch/kerberos/AliRoot) |
| \* | \* | \* | LDAP1  username/password | [https://git.cern.ch/reps/AliRoot](https://git.cern.ch/ldap/AliRoot)  <https://git.cern.ch/ldap/AliRoot> |
|  | \* | \* | CERN Public1 | <http://git.cern.ch/cernpub/AliRoot> |
|  |  | \* | Public1 | <http://git.cern.ch/pub/AliRoot> |
|  |  | \* | SSO Web2 | <https://git.cern.ch/cernpub/> |
|  |  | \* | Public Web2 | <http://git.cern.ch/pubweb/> |

(1): Client only.

(2): Browser access

R: Restricted, only username/password from anywhere in the world.

C: Cern, public anonymous read only from CERN IPs.

W: World, public anonymous read only from anywhere in the world.

Kerberos is by default installed in all SLC5 SLC6 machines, like lxplus. Public read only access is only available over HTTP, and for the repositories that requested it.

For further instruction how to set up Kerberos with CERN credentials please follow the link [Accessing CERN Linux machines via Kerberos](http://linux.web.cern.ch/linux/docs/kerberos-access.shtml)

**2.2 CERN Git Service FAQ**

CERN IT department created a list of [FAQ](https://cern.service-now.com/service-portal/faq.do?se=git-service) concerning the Git service. Please go through it to find all the usual issues related to CERN git service (requires CERN authentication).

**2.3 AliRoot access rules**

Due to Git internal organization CERN Git access rules are more restrictive than CERN SVN access rules.

**All authenticated accesses to GIT service(https URLs) require users to have commit rights on all or parts of the repository.**

**For browsing purpose please use only guest URLs (http not https)!**

This translates into:

1. SVN allows you checkout the project using restricted URLs even if you are not allowed at all to commit in AliRoot

          svn co <https://svn.cern.ch/reps/AliRoot/trunk> - it will ask you CERN credentials and it will allow the check out

2. Git will not allow users with no commit rights to use the following URL

          //DO NOT USE IF YOU DO NOT HAVE ANY commit RIGHTS  
         git clone <https://git.cern.ch/reps/AliRoot> - it will ask CERN credentials but it will deny the access

Instead of the URL above the users MUST use the guest URL such as

         git clone <http://git.cern.ch/pub/AliRoot>

         git clone <http://git.cern.ch/cernpub/AliRoot>

**2.4 AliRoot git web interface**

CERN Git central service offers 2 ways to access the web interface to AliRoot git repository.

1. Public interface gives access to anybody, including users with no CERN accounts or commit rights.

<http://git.cern.ch/pubweb/AliRoot.git>

2. Private interface that can be used ONLY by users that have commit rights to AliRoot. Note that this interface can't be used even if you have a valid CERN ALICE account but no commit rights. If you do not have commit rights please use the public interface.

<https://git.cern.ch/web/?p=AliRoot.git;a=summary>

**3. First-time Git Setup**

Git comes with a tool called "git config" that lets you get and set configuration variables that control all aspects of how Git looks and operates.

User specific configuration is stored in a ~/.gitconfig file. You can make Git read and write to this file specifically by passing the --global option.

1. Your identity - Please set your identity before committing to AliRoot repository.

Note that it is very important to set this information because the user.name is used by different automatic checkups such as Coverity.

To set this information globally use the git config command below.

$ git config --global user.name "YOUR\_CERN\_COMPUTING\_ACCOUNT"

$ git config --global user.email "YOUR\_EMAIL\_LINKED\_TO\_CERN\_COMPUTING\_ACCOUNT"

If you have more than one repository that you are committing and you are using different authoring information you can set it locally, per repository by removing --global flag.

Please remember to set this information for all your working instances.

2. Your editor - optional

$ git config --global core.editor emacs

3. Your diff/merge tool - optional

$ git config --global merge.tool vimdiff

    Git accepts kdiff3, tkdiff, meld, xxdiff, emerge, vimdiff, gvimdiff, ecmerge, and opendiff as valid merge tools.

4. Set Git to always rebase when pulling from a remote repository

$ git config --global branch.autosetuprebase always  (for all new branches

$ git config --global branch.master.rebase true  (for other existing branches, replace master with the branch you want to use)

If you do not want to rebase but you set it using git config you can use the command below to ignore rebasing

$ git pull --no-rebase

5. Push only the current branch you are working on

$ git config --global push.default simple

6. Color highlighting

$ git config --global color.ui true

$ git config --global color.status auto

$ git config --global color.branch auto

7. List the configuration

$ git config --global --list

For more information about how to further configure your git client please visit  
<http://git-scm.com/book/en/Customizing-Git-Git-Configuration>

**4. Commands**

For detailed explanation on the command please use :

     > man git command

**4.1 Browse commands**

|  |  |
| --- | --- |
| SVN | Git |
| Checkout trunk  svn checkout [http://svn.cern.ch/guest/AliRoot/trunk](http://svn.cern.ch/guest/AliRoot/trunk/) | git clonehttp://git.cern.ch/pub/AliRoot  (Note that "trunk" becomes "master")  Clone is done only once, checkout of the branches/tags is done inside the same working directory, one after the other.  If you want to checkout more than one branch/tag in parallel please follow the instructions from [Section 5](http://aliceinfo.cern.ch/Offline/node/2912/#ParallelGit). |
| Checkout a tag  svn checkout <http://svn.cern.ch/guest/AliRoot/tags/v5-04-72-AN> | Clone is done only once, checkout of the branches/tags is done inside the same working directory, one after the other  git clonehttp://git.cern.ch/pub/AliRoot  cd AliRoot  1. Checkout in "HEAD detached mode". When HEAD is detached, it points directly to a commit—instead of indirectly pointing to one through a branch. You can think of a detached HEAD as being on an unnamed branch.  git checkout v5-04-72-AN  git checkout v5-04-73-AN  A "git branch" will show  $ git branch \* (detached from v5-04-72-AN)   master 2. Checkout into a local branch            git checkout -b <new\_branch\_name> <tag>  git checkout v5-04-72-AN-local v5-04-72-AN  or to just keep it simple, use the same name for the branch as for the tag  git checkout -b v5-04-72-AN v5-04-72-AN    If you want to checkout more than one branch/tag in parallel please follow the instructions from [Section 5](http://aliceinfo.cern.ch/Offline/node/2912/#ParallelGit). |
| Checkout a branch  svn checkout [http://svn.cern.ch/guest/AliRoot/branches/TPCdev](http://svn.cern.ch/guest/AliRoot/%28trunk/branches/tags%29) | Clone is done only once, checkout of the branches/tags is done inside the same working directory, one after the other  git clonehttp://git.cern.ch/pub/AliRoot  cd AliRoot  git checkout TPCdev  git checkout AN-201306-15  If you want to checkout more than one branch/tag in parallel please follow the instructions from [Section 5](http://aliceinfo.cern.ch/Offline/node/2912/#ParallelGit). |
| List tags and branches  svn list [http://svn.cern.ch/guest/AliRoot](http://svn.cern.ch/guest/AliRoot/%28trunk/branches/tags%29) | List tags and branches on local repository  git tag/branch  List tags and branches on remote repository  git remote show *<remote>*  <remote> is usually *origin* |
| svn blame *file* | git blame *file* |
| svn info | There is no complete git equivalent of svn info command.  1. git log - will show on the first line the current branch in the working directory  2. [git-info](https://github.com/gitbits/git-info/) contribution script - download all 3 scrips: git-info, git-pager and git-editor, make them executable and add them to $PATH |

**4.2 Update commands**

|  |  |
| --- | --- |
| SVN | Git |
| Status of the working directory  svn status | In Git case we find 2 use cases  1. Status of the local repository  git status  2. Status of the local repository compared to a remote  git fetch  git log  git merge (if you want to merge to local repository) |
| Updating working directory  svn update | There are 2 ways to integrate changes: [merge](http://git-scm.com/book/en/Git-Branching-Basic-Branching-and-Merging) or [rebase](http://git-scm.com/book/en/Git-Branching-Rebasing)  1. Merge - the changes are merged directly creating a new commit with a new history.  git pull or  git fetch && git merge  (the last one will allow to see all files that were touched on the central repository)  2. Rebase - A rebase rolls back your history to the point where you forked from the remote, applies the remote changes, and then reapplies your local changes commit by commit. This rewrites history and should therefore only be used on unpublished branches.  git fetch && git rebase or  git pull --rebase (preferred method)  To better understand the difference between the usage of one or the other:  <http://gitolite.com/tips/git-pull--rebase.html>  <http://stackoverflow.com/questions/804115/git-rebase-vs-git-merge>  <http://stackoverflow.com/questions/3357122/git-pull-vs-git-fetch-git-rebase> |
| Switching between branches  svn checkout [http://svn.cern.ch/guest/AliRoot/branches/AN-2013-05-16/](http://svn.cern.ch/guest/AliRoot/%28trunk/branches/tags%29)  svn switch AN-2013-05-24 | git clonehttp://git.cern.ch/pub/AliRoot  cd AliRoot  git checkout AN-2013-05-16  git checkout AN-2013-05-24 |
| Committing your changes  svn commit | !Before commit always *pull/fetch* the changes from central repository!  1. Commit to local repository  Git allows to "add" changes to a prestage area. Therefore you always need to add the changes to the index before committing them.  git add *ModifiedFile*  git commit  or  git commit -a  2. Push the changes to remote repository  git push  The command above will push all local repository to central repository.  If only the current branch has to be pushed then use  git push --set-upstream origin current\_branch\_name  Another option is through  git config  push.default *upstream*  *push\_value =* [*nothing/matching/upstream/simple/current*](http://nothing/matching/upstream/simple/current) |
|  | git add *file* |
| svn rm *file* | git rm *file* |
| svn mv *file* | git mv *file* |
| svn revert *rev* | git revert *rev*  If a commit has been made somewhere in the project's history, and you later decide that the commit is wrong and should not have been done, then *git revert* is the tool for the job. It will undo the changes introduced by the bad commit, recording the "undo" in the history.  git checkout *rev*  If you have modified a file in your working tree, but haven't committed the change, then you can use *git checkout* to checkout a fresh-from-repository copy of the file.  git reset --hard *rev*  If you have made a commit, but haven't shared it with anyone else and you decide you don't want it, then you can use git resetto rewrite the history so that it looks as though you never made that commit. |

4.3 Advanced commands

|  |  |
| --- | --- |
| SVN | Git |
| Branching  svn copy [http://svn.cern.ch/guest/AliRoot/branches/TPCdev http://svn.cern.ch/guest/AliRoot/branches/TPCdev\_new](http://svn.cern.ch/guest/AliRoot/%28trunk/branches/tags%29)  svn switch [http://svn.cern.ch/guest/AliRoot/branches/TPCdev\_new](http://svn.cern.ch/guest/AliRoot/%28trunk/branches/tags%29) | git branch TPCdev\_new TPCdev  git checkout TPCdev\_new  Make changes and if needed push the branch to central repository  git push |
| Merging  svn merge [http://svn.cern.ch/guest/AliRoot/branches/AN-2013-05-16/](http://svn.cern.ch/guest/AliRoot/%28trunk/branches/tags%29) | git merge AN-2013-05-16 |

4.4 Storing passwords

Git can store your password in a secure way using a git-credential-helper. As of git 1.8 these are

* cache
* gnome-keyring
* osxkeychain
* wincred
* netrc (not secure, password in cleartext)

To get the gnome helper (on most current Linux distros) you will need to get git and build the helper yourself:

git clone [https://github.com/git/git.git](https://github.com/git/git.git" \o "https://github.com/git/git.git)

cd git/contrib/credential/gnome-keyring

make

sudo cp git-credential-gnome-keyring /usr/local/bin

git config --global credential.helper /usr/local/bin/git-credential-gnome-keyring

On OSX machines, the osxkeychain helper is by default installed, to use it do:

git config --global credential.helper osxkeychain

In a terminal environment on Linux you can use the 'cache' authentication helper that is bundled with git 1.7.9 and higher:

git config --global credential.helper 'cache --timeout 3600'

After enabling credential caching any time you enter your password it will be cached for 1 hour (3600 seconds).

**5. Git tips and best practices**

Git offers some extra features that can be very useful during development

5.1 git stash [[3](http://git-scm.com/book/en/Git-Tools-Stashing)]

If you are working on a branch, get to a state, but do not want to commit you can save the status of the working repository and than switch to another branch.

5.2 git cherry-pick [[4](http://wiki.koha-community.org/wiki/Using_Git_Cherry_Pick)]

If you want to test a feature someone has developed, but it only exists in a remote branch which is woefully out of date.  With cherry-pick you can select and apply only the commits you are interested on.

**6. Checking more than one branch/tag in parallel**

Git best practices advise to use only one working directory, therefore, clone once and then checkout the branches/tags one by one, one after the other one.

But because AliRoot is using Git also as a software distribution system the users need to have in parallel more than one tag/branch.

The problem has 2 solutions:

6.1 Clone the repository for each tag/brach

git clone [http://git.cern.ch/pub/AliRoot](http://git.cern.ch/pub/AliRoot" \o "http://git.cern.ch/pub/AliRoot) AliRoot-1

cd AliRoot-1

git checkout v5-04-71-AN

And then again

git clone [http://git.cern.ch/pub/AliRoot](http://git.cern.ch/pub/AliRoot" \o "http://git.cern.ch/pub/AliRoot) AliRoot-2

cd AliRoot-2

git checkout v5-04-72-AN

This procedure will cost 3.2 GB x numbers of tags/branches you checkout

The second remote clone can be avoided by cloning locally only for the first branch and then clone the local repository for the second branch.

git clone [http://git.cern.ch/pub/AliRoot](http://git.cern.ch/pub/AliRoot" \o "http://git.cern.ch/pub/AliRoot) AliRoot-1

cd AliRoot-1

git checkout v5-04-71-AN

And for the second branch

cd ..

git clone AliRoot-1 AliRoot-2

cd AliRoot-2

At this point we have to clones, but the second one has as origin point the first one, so we need to change the origin to central repository

git remote rm origin

git remote add origin [http://git.cern.ch/pub/AliRoot](http://git.cern.ch/pub/AliRoot" \o "http://git.cern.ch/pub/AliRoot)

git checkout v5-04-72-AN

6.2 Using [git-new-workdir](http://nuclearsquid.com/writings/git-new-workdir/) contribution

Every git installation comes with a contribution "contrib" directory that contains different useful extension scripts.

For Ubuntu the "git-new-workdir" script can be found here

/usr/share/doc/git/contrib/workdir/git-new-workdir

The location can differ depending of git version or operationg sistem. Use "locate" command to find the actual installation point.

The "git-new-workdir" script needs to be runnable and added to your $PATH. One the script is in the $PATH you can:

git clone [http://git.cern.ch/pub/AliRoot](http://git.cern.ch/pub/AliRoot" \o "http://git.cern.ch/pub/AliRoot) AliRoot-1

mkdir AliRoot-2

git-new-workdir AliRoot-1 AliRoot-2

cd AliRoot-2

git checkout "v5-04-72-AN"

This procedure will use 3.2 GB for the initial clone plus 1.1 GB for each checkout.

**7. Useful link**

A list of useful link when migrating form SVN to Git

* [Git Best Practice (CERN presentation](http://indico.cern.ch/conferenceDisplay.py?confId=288437))
* <http://git.or.cz/course/svn.html>
* [http://people.gnome.org/~newren/eg/git-for-svn-users.html](http://people.gnome.org/%7Enewren/eg/git-for-svn-users.html)
* <https://git.wiki.kernel.org/index.php/GitSvnCrashCourse>
* <http://git-scm.com/book>
* <http://refcardz.dzone.com/refcardz/getting-started-git>
* <http://root.cern.ch/drupal/content/git-tips-and-tricks>