Git

```
  Update (15/03/2012): Port 2222 is depricated. All services run
additionally on Port 22
  Update (13/03/2012): More than 5 wrong authentications yield into a 10
minute ip-address ban
```

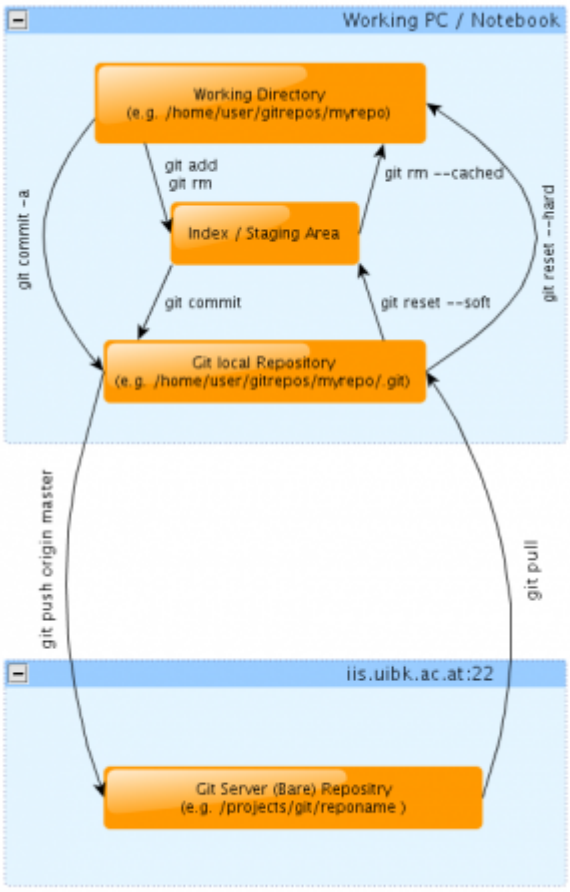
Git is an extremely powerful and flexible revision control system, and using it effectively requires adherence to conventions.

IIS Repository Policy

- A distinct tree / a repository - is maintained for each independent (software, paper, …) project.
- For each repository, the mainline (master) tree is hosted on the IIS server.
- Contributors develop locally and maintain their own trees, and push (or request to pull) only generally-useful, tried-and-proven patch sets into the main tree.
- Where appropriate, multiple external developers can exchange patches among each others before committing to the mainline.
- Branches are used to maintain multiple releases simultaneously while developing new features in the trunk. As long as we will not generally have formal releases, there will be little or no need for branches.

Please adhere to basic guidelines for Editing and Committing.

Basic Workflow



Retrieve an IIS Git tree for you to work on:

```
git clone ssh://iis.uibk.ac.at/projects/git/PROJECTNAME
```

PROJECTNAME will typically include a path portion.

Prepend the hostname with USERNAME@ if your Git username does not match your local username.

Now make your edits. To see the status of your files with respect to your repository, do:

```
git status
```

For each reasonable unit of changes, tell Git that you want to keep it, and then commit it to your local tree:

```
git add FILENAME
git commit -m "brief documentation of your changes"
```

Each self-contained, fully-functional set of changes that you want to make public should be pushed upstream to the IIS master tree:

```
git push
```

At this point you are free to delete your local Git tree.

In the meantime, you can retrieve updates from the IIS master tree with:

```
git pull
```

For more information, see the Git Reference and Everyday Git.

A `git pull` will automatically merge any edits other people have pushed in the meantime. However, this will fail if such edits happened in the same section of a file as the changes you are trying to push. In this case, `git pull` will signal a merge conflict, which needs to be resolved by hand.

Here you may find a nice tutorial on how you can do merging by using vimdiff (you should be able to use vim for using it).

Creating a New Git Project

To create a shared Git repository, initialize it on the lab server. Then, clone and populate it.

To create your personal Git repository for your own, local use only:

```
cd PROJECTROOT
git init --shared
git add .
git commit -m "initial import"
```

Understanding Git

Here is an excellent and brief explanation of Git's internal representation and how it supports user-level interactions. Very useful to understand Git.

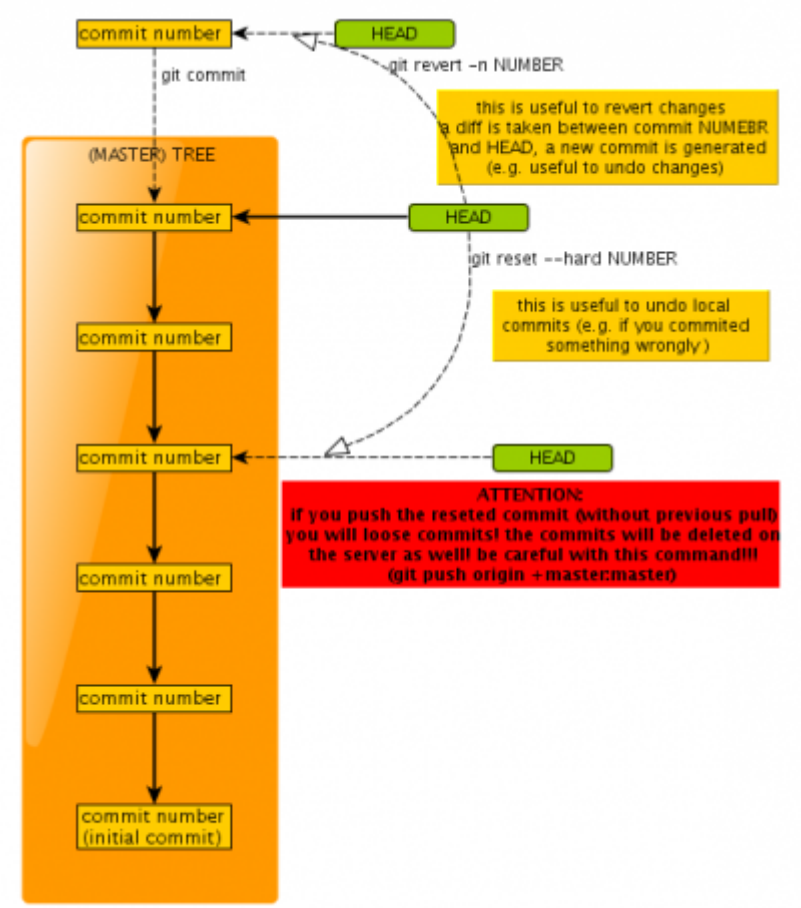
Two key distinctions from Subversion (SVN) are

- the *staging area* (“index”) where you define the changes to be commited. (SVN commits directly from the working directory.)
- the *distributed* nature. Commits are done to the local clone of the repository. Sharing your changes requires explicit action, e.g., `git push`. (SVN commits directly to a central repository.)

More Advanced Hints

Git Revisions

The following diagram is an example 'commit' view of a master tree (w.o. branches). it describes the difference between revert and reset.



Moving or Renaming a Git Repository

A Git repository does not know its own name or location; it is simply identified by its location in the filesystem. It can be moved or renamed ad libitum.

To keep any cloned copies in sync, you have essentially two options:

- Commit and push everything before the move, delete the clone, move the repo on the server, and create a fresh clone.
- Point an existing clone to the updated location: From the root directory of the cloned tree, issue

  ```
  git remote set-url origin
  ssh://iis.uibk.ac.at/projects/git/SUBDIR/REPONAME
  ```

  See here and `man git-remote` for more information.

Textmode Tool For Git

a nice (commandline) tool for viewing commits, logs, diffs and other changes is tig. to install tig on a debian based machine

```
aptitude install tig
```

change into the git working directory and start

```
tig
```

Patches

Generate Patches and use Peer-to-peer patch exchanges [1)]

```
git format-patch --cover-letter -o some-dir
d8a285c8f83f728be2d056e6d4b0909972789d51..9202ec15da36ca060722c363575e0e390d
85fb71
# this is equivalent to, this is the short form
git format-patch --cover-letter -n -o some-dir d8a28..9202e
```

Where d8a28 was the last commit before you started hacking and 9202e is the current head, meaning the commit ID of your latest commit.

For renaming files add “-M” to the git-format-patch arguments then patches wont create removals and adds for a simple rename.

Sending Patches:

```
git send-email --no-chain-reply-to --from "My Name <my.name@uibk.ac.at>" --
to recipient@domain some-directory/
```

Ssh Config

You could also create an ssh config file in your home-directory to shorten the git commands: The ~/.ssh/config file could look like:

```
Host iis
    HostName iis.uibk.ac.at
    Port 22
    User username
    #IdentityFile  ~/.ssh/PRIVATEKEYFILE
```

The IdentityFile line is only for ssh key auth necessary.

The commands then would look like

```
git clone ssh://iis/projects/git/projectname
```

search?q=git%20Usage&btnI=lucky

1)
example taken from http://linuxwireless.org/en/developers/Documentation/git-guide