AQ

GIT

Distributed version control system
for distributed, non-linear workflows

Git is a version control system created by Linus Torvalds to manage the development of the **Linux kernel**

... Because SVN is not working for me

- Linus Torvalds

WHAT'S WRONG WITH SVN?

- Single remote repository only
- Folder based branches
- Inefficient merging
- No file ignore templates

WHAT'S RIGHT WITH GIT?

- It's just local
- Every local repo could be a remote
- Fast branching and tagging (not a copy of the repo)
- Advanced interactive merging and *cherry-picking*
- Distributable `.ignore` and configuration files

GETTING STARTED

COMMAND LINE INSTALL:

- Official build
- Included in cmder

GUI:

- Included in WebStorm / PHPStorm and Visual Studio
- Source Tree (lacks built-in merge tool)

KEY CONCEPTS

- Working copy: your local repository.
- Index: a compressed list of files in the repo
- Commit: a set of metadata for each change introduced into the repository. It's a complete snapshot of the repository.
- Tag: a human readable label pointing to a specif commit
- Branch: a timeline of development. The default timeline is conventionally called *master*
- Merge: the action of syncing a branch with another

- Remote: a remote repository (usually a git server)
- Clone: creates a local copy of a remote repository
- Pull: downloads remote changes and merges them with your local repository
- Push: uploads local changes and merges them with remote repository

PLEASE NOTE

Git is a distributed VCS. You will always **work locally** and then **sync** with remotes.

Your working copy may have **local branches and tags** which are not present in the remote server and vice-versa!

AQ REMOTE SERVER

AQ has a private remote server powered by GitLab

GitLab

https://git.aquest.it/

MAIN FEATURES:

- Browsable sources
- Private / restricted access projects
- Working groups
- Per project issues, wikis, milestones and attachments

You should have received your personal username/password access. If not:

ASK THE SYS ADMINS

To work with repos you should prefer a **SSH authentication** access.

- Follow this guide to create your SSH key and add it to GitLab
- Follow this reference to setup SSH access in SourceTree

Note: *you need a key for every machine you're connecting from.*

REPO MANAGEMENT

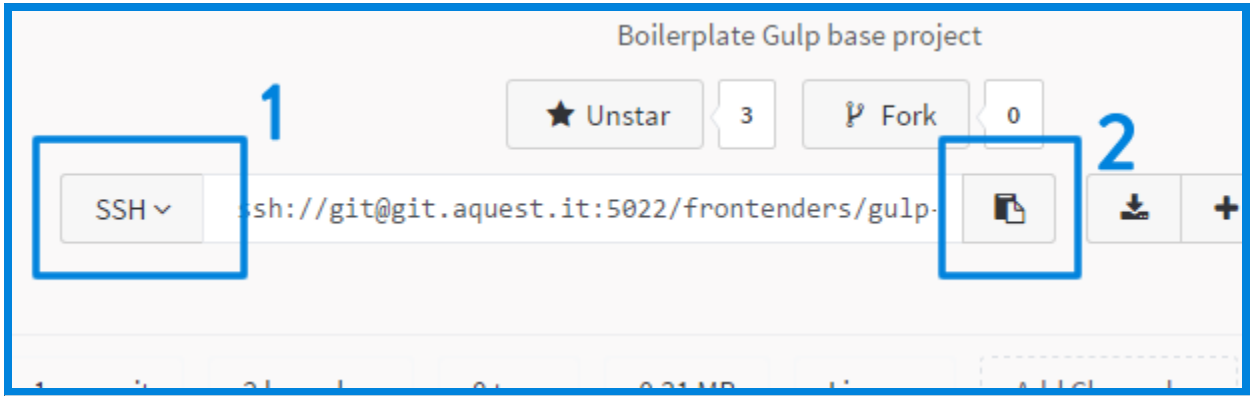
the CLI way...

CREDENTIALS AND LINE ENDING

Ensure your username and email are correct (they will be used in logs. Also let git manage EOL for you

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe+git@aquest.it
$ git config --global core.autocrlf true
```

CLONE FROM AN ALREADY EXISTING REMOTE REPOSITORY




```
#clone a project in the gulp-boilerplate folder

$ git clone ssh://git@git.aquest.it:5022/frontenders/gulp-boilerplate.g
$ cd gulp-boilerplate
```

SETUP A NEW LOCAL PROJECT AND SET DEFAULT REMOTE REPO

```
$ cd my-project
$ git init
$ git add remote origin ssh://git@git.aquest.it:5022/my-project.git
```

Since there's no 1 to 1 relation between local and remote repo, you may add more than one remote and sync them independently.

```
$ git add remote github git@github.com:username/my-project.git
$ git add remote customer-repo ssh://git@git.mycustomer.com/my-project.
```

Note: by convention both commands will create a `master` branch

WORKING WITH FILES

CHECK REPO STATUS AND LOGS

To check what's changed in your local repo use `status`.
Use `log` to see previous commits

```
$ git status #check for modified files
$ git log #commit logs
```

ADD AND COMMIT FILES

In order to update your repo with new file changes you need to:

1) Add *(stage)* them to the index
2) Commit them to the repository

```
$ git add path/to/file.html #stage a single file

$ git add . #stage every modified / added / removed file

$ git commit -am "my comment" #commit staged files (a) with a comment (
```

IGNORING FILES

Git introduces a special file `.gitignore` to list ignored file on a per-project level. It's a simple text file listing wildcard path matches

```
.DS_Store
.tmp/ #keep the folder and ignore it's contents
dist/*.min.js #wildcard matching
```

Note: By default git won't sync / add empty folders. If you need an empty folder place an empty `.gitkeep` file in it

DEAL WITH ERRORS

There're different strategies to deal with errors...

DON'T PANIC!

ROLLBACK MODIFIED FILES

If you modified some files and want to rollback use `reset --hard`

```
#rollback all files to the latest committed status
$ git reset --hard HEAD
```

ROLLBACK ALREADY COMMITTED FILES

If you want to go back to a previous version of your project use `revert`. `revert` will create a new commit thus advancing your commit timeline.

```
#rollback all files to a specific commit hash
$ git revert 85431554a2c932310a6eed0f1ef6b0bfdad70693

#rollback to previous commit
$ git revert HEAD~1
```

Note: `revert` is applied to every indexed file in the repo.

ROLLBACK A SPECIFIC FILE

To rollback a specific file use `checkout`

```
#rollback a specific file to a specific commit hash
$ git checkout 85431554a2c932310a6eed0f1ef6b0bfdad70693 path/to/file.ht

#rollback to previous commit
$ git checkout HEAD~1 path/to/file.html
```

REFERENCES

- Reset, Checkout, and Revert

BRANCHES AND TAGS

BRANCHES

Branches are independent timelines of your project. To create / move between branches use `checkout`

```
#create a new branch
$ git checkout -b my-branch

#switch to a new branch
$ git checkout master

#remove a local branch
$ git checkout -d my-old-branch
```

Note: remember that Git works locally, at this stage branches are just local to your repository

MERGE BRANCHES

To update one branch with changes from another branch use `merge` or `rebase`

- `merge` adds changes from another branch creating a new commit
- `rebase` rolls back to the branches' common commit, updates target branch and re-applies file changes

```
# merge branches
$ git checkout master
$ git merge my-development-branch

# update development branch with fixes from master
$ git checkout my-development-branch
$ git rebase master
```

TAGS

Tags are *bookmarks* to a specific commit of the project

```
#list all tags
$ git tag

#create a new named tag
$ git tag -a here-it-works -m "Until this point every works fine!"

#create a new version tag
$ git tag -a 1.0.0 -m "stable release"
```

Note: like branches, at this stage branches are just local to your repository

REFERENCES

- Using branches
- Merging vs Rebasing

To get the best from Git workflow you should always work on development branches, merge with master just stable features and tag them

ENTERS GIT-FLOW

git-flow is a set of scripts to help in branch based git workflow

Installation in windows may be tricky. If you already installed cmder (which I strongly suggest to) you have to follow this instructions.

Once installed use `git flow` to setup and run commands

```
$ git flow init #init the environment
$ git flow help #show contextual help
$ git flow feature start [feature-name] #a sample command
```

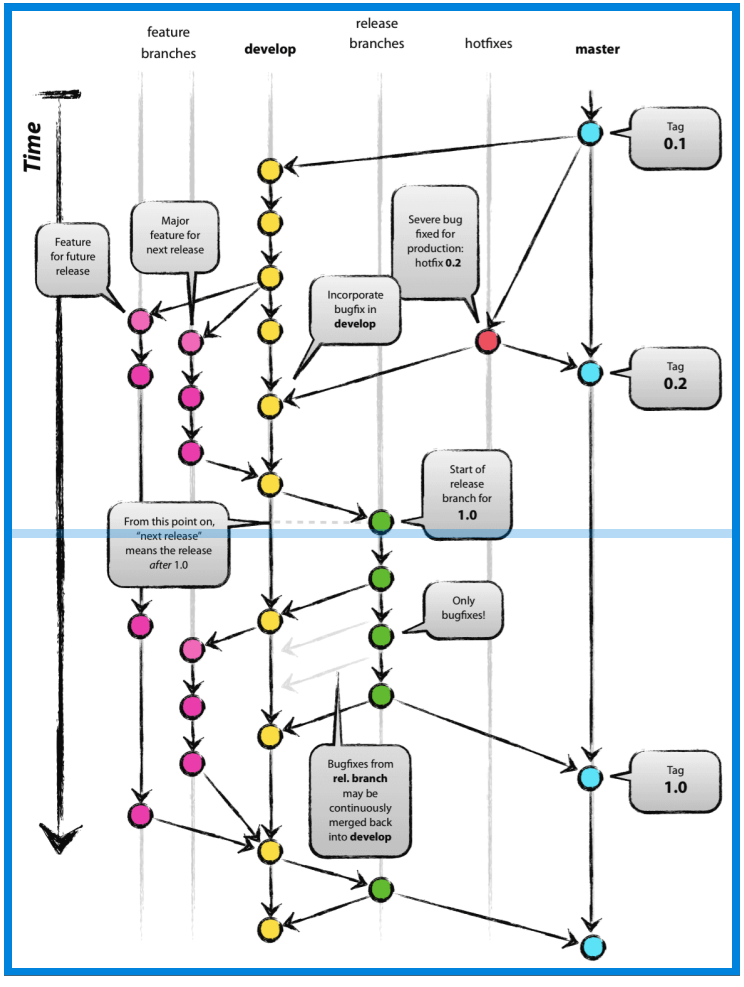
There're 2 **default branches**.
You should **never** work directly on them

- `master` branch is the most stable version of the codebase
- `develop` branch is the collective development version

There're 3 main **namespaced** types of working branches

- `feature/*` : a in-development discrete part of the application.
- `hotfix/*` : bugfix management
- `release/*` : preparation stage before releasing a new stable version

HOW IT WORKS...



command cheatsheet

WORKING WITH REMOTES

Until now everything happened on your local repo.
Let's collaborate!

PUSH AND PULL

By convention the main remote repository is called `origin`.
To sync with remote updates use `pull`

```
#switch to master branch
$ git checkout master

# sync with branch master of the remote labeled as "origin"
$ git pull origin master
```

To upload your changes use `push`

```
#switch to master branch
$ git checkout master

# upload changes to branch master of the remote labeled as "origin"
$ git push origin master
```

Since you may have **multiple remotes**, you may sync discrete remotes

```
$ git push origin master #sync with AQ git server for backup

# ...after some commits and branching

# sync something more stable with the client's shared server
$ git push external master
```

You can also push and pull tags

```
# create a tag
$ git tag -a 1.0.0 -m "first stable release"

# push tags
$ git push origin --tags

# sync tags
$ git pull origin --tags
```

REMOVE REMOTE TAGS AND BRANCHES

To remove a remote branch push its name prefixed with `:`

```
# remove a tag
$ git push origin :1.0.0

# remove a branch
$ git push origin :my-branch
```

DEMO TIME!

https://git.aquest.it/

THANKS