Git Usage

Which git repository to clone

The Gitlab Mirror

`git@gitlab.com:tikiwiki/tiki.git`
An automated mirror. **This is what you should clone if you want to do pull requests** on https://gitlab.com/tikiwiki/tiki

No need to ask anyone, but you DO need to upload you ssh key to https://gitlab.com/profile/keys

git.tiki.org

`git@git.tiki.org:tiki.git`
The main repository is git@git.tiki.org:tiki.git. You need to have your public ssh-rsa key registered to that server if you are going to merge the accepted pull requests from the Gitlab mirror. Ask luciash <luci@tiki.org> or Fabio <fabio.montefuscolo@gmail.com> if you need that direct access.

Getting the source

Shallow clone

Tiki development has been going on for nearly 20 years, and the code base is huge. For these two reasons the repository has a huge size and a regular clone will take a lot of time and disk space.

But it is possible to create a shallow clone, which will bring just the last source code version.

Shallow clone of every branch

```
git clone --depth 1 --no-single-branch git@gitlab.com:tikiwiki/tiki.git tiki
```

This is quite fast, and uses about 292M of disk space. Probably best for all developers.
`git branch -a` works like most people expect and it's easy to switch branch.

If the purpose of the clone is Tiki development, including master and other branches, it is not needed to clone the entire repository, buy you my want to use --depth 100 so you have some useful history. (this will inflate the size to about 455M however.

Shallow clone, single branch

```
git clone --depth 1 --branch master git@gitlab.com:tikiwiki/tiki.git tiki
```

The smallest option, but you can't use `git branch -a` like you normally would. instead you need to use `git ls-remote`.
To add another branch, you need to use the variant below.

This option is best to checkout on production servers. It uses about 187M of disk space

Shallow clone, more than one branch

```
git clone --depth 1 --branch master git@gitlab.com:tikiwiki/tiki.git tiki git remote set-branches --add origin '21.x' git fetch --depth 1 #REALLY important on the first fetch, otherwise that whole branch will be fetched with all history. That will use up 1.4G and take a really long time. #For normal operations you can just git fetch normally, it won't un-shallow the branch.
```

It now uses about 187M of disk space, which is barely more than a single branch. This option is best if you rely on expensive data connections and/or have very limited disk space.

Full clone

Full clone

```
git clone git@gitlab.com:tikiwiki/tiki.git
```

This is quite slow, and uses about 3.1G of disk space. However subsequent operations are not any slower, and you have full local history. This is best if you have the time for the initial clone, and you need to work completely offline.

Full clone with folders per branch

`git clone --reference`

Full git repository is 3.1G. That's because the whole history is stored locally. Since Tiki setup depends on a lot of libraries, changing branches in a single repository might be impractical, and having one repository for each directory would consume a lot of redundant disk space.

To solve that, start by having a local mirror repository that will be used as reference for all other repositories. This way you can have several repositories that share the local object database. It's like doing a pre-cache of the repository:

Create a local mirror

```
git clone --mirror git@gitlab.com:tikiwiki/tiki.git tiki-reference
```

(This will take some time, have a coffee).

Supposing a developer wants to clone also the branch 21.x, but without cloning the entire repository, the following command should be used.

Checkout against a reference

```
git clone --branch=21.x --reference=tiki-reference git@gitlab.com:tikiwiki/tiki.git tiki21x
```

Repeat for other branches.

Now just commit and push normally, and everything should work fine.

Key issues

In practice, git is often based on ssh, which may be confusing if you've never used it before. Among other things, it will ask to accept the key of the remote server the first time you use it.

Accepting the key of the remote server.

```
clone --mirror git@git.tiki.org:tiki.git tiki.reference Cloning into bare repository 'tiki.reference'... The authenticity of host 'git.tiki.org (172.104.234.244)' can't be established. ECDSA key fingerprint is SHA256:bsX1VZ1KCXiUlequpZc6f+JGBG0xO4HpUSyYnVrXbMs. Are you sure you want to continue connecting (yes/no)? yes Warning: Permanently added 'git.tiki.org,172.104.234.244' (ECDSA) to the list of known hosts. git@git.tiki.org's password:
```

Repository status

`git status`

You can read about changed files, untracked files and how far is your local repository from the remote repository by using the `git status` command. More information can be found by typing `git help status`.

Status

```
git status
```

Status with ignored files

`git status --ignored`

The command below will list all ignored files present on the working copy. This is useful when trying to commit a file that should be on repository, but it is ignored by a rule present in `.gitignore` file.

Ignored files status

```
git status --ignored
```

Status of untracked files

`git status -u`

Sometimes, depending the task, it is possible that working copy contains a lot of untracked files. But it is possible to check working copy status excluding those files.

Status excluding untracked files

```
git status -u no
```

There are other occasions when there is a need to check all untracked files, including files on sub-directories.

Status of all untracked files

```
git status -u
```

Reading history

To read the commit log, container authors, data and commit messages, you can use the command `git log`.

Log

```
git log
```

Filtering history

`git log --author --since`

It is possible to filter logs by author and date, like the snippets below demonstrate.

Filter by user

```
git log --author=jonnybradley # --author can be used multiple times git log --author=jonnybradley --author=drsassafras
```

Filter by date

```
git log --since=2018-03-01 # --since is the start date, and --until is the end date git log --since=2018-03-01 --until=2018-04-31
```

All together

```
git log --since=2018-03-01 --until=2018-04-31 --author=jonnybradley --author=drsassafras
```

For more options, try `git help log`

Adding changes and files

`git add`

Before committing, the changes you did should be marked as added. So, when you tell git to commit your changes, only files you marked as added will be committed.

Add changes

```
echo 'My new super file' > src/foo.txt echo 'Other super file' > src/bar.txt git add src/foo.txt src/bar.txt
```

It is possible to add all changes from a folder at once. It also works to add changes from entire working copy, but it is not a good practice.

Add changes

```
echo 'My new super file' > src/foo.txt echo 'Other super file' > src/bar.txt git add src git commit -m '[NEW] Add new super feature to Tiki'
```

Removing files from Git

`git rm`

Removing files

```
git rm src/foo.txt git commit -m '[FIX] Removed unused file'
```

Setting up global .gitignore

This from Stack Overflow:

Ignoring files

```
git config --global core.excludesfile '~/.gitignore'
```

the to ignore all PhpStorm project settings, fo instance add this to ~.gitignore

~/.gitignore

```
/.idea/
```

Excluding personal files from a specific git repository

Sometimes you want git to ignore files on a specific local repository, not on all of your local repositories (where you would use a global .gitignore). But you also don't want to add those files into the repository .gitignore, because those are personal files (like files related to your code editor, for example). In that case, you can edit the `.git/info/exclude` file of your repository. This file works like a .gitignore file, but will not be committed to the repository.

See for more information: https://stackoverflow.com/a/22906950/751791

Committing changes

`git commit`

After all new or changed files are marked to be committed by using `git add`, or files were removed by using `git rm`, it is time to commit these changes to your local repository.

Committing changes

```
git status git commit -m '[NEW] Descriptive message about my changes'
```

There are occasions a developer commits changes from someone else. In this case, it is possible to give the commit the correct author identity.

Committing changes of someone else

```
git commit --author=someoneelse@example.com -m '[FIX] Options list for editable icon preference'
```

When there are changes just on versioned files, there is shortcut to commit these changes at once, by adding `-a` parameter on git commit.

Different author

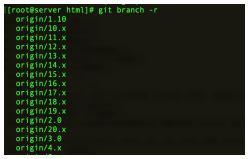
```
git commit -a -m '[NEW] Descriptive message about my changes'
```

Working with branches

Apart from master (trunk in SVN) Tiki has several branches. Some are permanent, used in parallel (for LTS and stable still supported Tiki versions), some are experimental and some are temporarily created by developers/contributors for merge request.
You can check the available branches by using the branch command;

Display all the branches

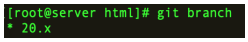
git branch -r



On a development environment you may use several Tikis and several branches.
You can check on which branch you are by executing the branch command from inside a Tiki directory;

Display the working branch

git branch



Creating branches

`git branch`

Creating branches is easy and fast on Git. It is a good idea to have different branches for tasks, features or experiments and keep master branch clean, so you can fetch updates regularly without dealing with conflicts.

Create branch

create the branch git branch task-icon-picker # and then switch to branch git checkout task-icon-picker

There is a shortcut to create a branch and then switch to it.

Create branch

git checkout -b task-icon-picker

If this local branch is needed for other team mates, it is possible to make it available on remote repository.

Create branch

git push -u origin task-icon-picker

Let's suppose you are working on branch *bugfix-foo*, and you want update your branch with branch master changes.

Merging branches

git checkout bugfix-foo # switch to branch git merge master # bring master changes to current branch

Switching branches

If you checked out a single shallow branch (e.g. 21.x) as described above you may get errors like this if you try, for instance `git checkout origin/22.x`:

> Git: cannot checkout branch - error: pathspec '...' did not match any file(s) known to git

Then try this (note, i did something else and ended up checking out the whole 3.6GB tiki repo! 🖼)

Switch branch having only checked one out

```
git remote set-branches --add origin '22.x' git fetch --depth 1
```

Sending work to remote repository

`git push`

To send your new commits from your local repository to the remote repository, just use the command `git push`. The command below will send all changes on local branch to the remote branch on remote repository.

Pushing changes

```
git push
```

Fetching updates

`git pull`

It is a good practice to stay always up to date with remote repository your team is using. This way you avoid a lot of conflicts. To get remote changes and bring to local repository and working copy, use the command `git pull`.

The command below will fetch updates from remote repository.

Fetching changes

```
git fetch
```

After fetching changes, the local repository and branch should be updated with it. Supposing the branch to be updated is the `master`, and the current local branch is also `master`, it is possible to apply changes with the command below.

Applying changes

```
git merge origin/master
```

There is a shortcut to fetch and apply changes on current branch, like below.

Fetching and applying

```
git pull
```

Backporting fixes to previous releases

See: Git cherry-pick

Git undo

See: https://stackoverflow.com/questions/927358/how-do-i-undo-the-most-recent-local-commits-in-git