Advanced Git

Resources

Some resources for learning GIT:

- Interactive Git basics
- Get into a workflow (important when working in a team!)
- One of the more popular workflow methodologies is Git-flow - Here's also a 15 minute video intro
- Also take a look at the GIT Usage page in this wiki.

Tools

- If your team picks up Git-flow (mentioned above) as the standard, check out this shell envancement that will help you stick to the program.
  - Easiest way to get it is "apt-get install git-flow", in case you are on Debian/Ubuntu/Mint or yum install gitflow on RHEL and friends.
  - **Good starter guide on Getting Started with git-flow**
- SourceTree is a client for win and mac. TortoiseGit is also popular but last time I checked it (2011) was a bit slow, lacked features.
- Our current setup hosts the central repo on Gitolight, Help me persuade Igor to more us to GitLab :-)

Git-flow init

Before doing any git flow feature * you should make git flow init

```
git flow init
Which branch should be used for bringing forth production releases?
  - CBnode-2.1
  - master
Branch name for production releases: [master]

Which branch should be used for integration of the "next release"?
  - CBnode-2.1
Branch name for "next release" development: [develop] CBnode-2.1

How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
```

Git-flow cheat sheet

Make sure you are in the latest version

```
git checkout CBnode-2.4
git pull
```

Create new feature

```
git flow feature start FEATURE_NAME - branches from "develop" (usually a branch called
"dev" at CloudBand).
git flow feature publish FEATURE_NAME - makes sure the feature is pushed up to origin
(usually the central repo) and the other team members can check it out.
git flow feature finish FEATURE_NAME - merges the feature back into develop and if
successful, removes the branch.
```

Please note that many times we branch and merge to a version branch like CBNode-1.5.2 or CBNode-1.6 - you can't use git flow feature * with that, sadly. Work carefully and try to keep with the workflow.

(for more see tips and tricks below)

Shell stuff

Git Aliases for bash

```
alias ga='git add '
alias gb='git branch '
alias gc='git commit'
alias gd='git diff'
alias get='git '
alias gk='gitk --all&'
alias go='git checkout '
alias got='git '
alias gs='git status '
alias gx='gitx --all'
alias logit='git log --pretty=format:"%h %ad | %s%d [%an]" --graph --color --date=short'
```

Prompt, Auto-completion and other fun stuff

for more aliases and tab-completion, I suggest you install bash-it. I used it to fix a nice prompt theme (dynamic $PS1 on steroids) that tells me which branch I'm on and if it's clean.

If you want my repo (with my prompt and fixes) then clone from here instead, or just pull on it to get the extra goodies.

If you want just the prompt without a lot of fussy aliases and plugins, I recommend this

Git aliases for ~/.gitconfig

```
[alias]
   co = checkout
   ci = commit
   st = status
   br = branch
   hist = log --pretty=format:\"%h %ad | %s%d [%an]\" --graph --date=short --color
   aliases = config --get-regexp alias
   new = !sh -c 'git log $1@{1}..$1@{0} "$@"'
```

More aliases, suggested by Ido Sekely, the stg-* are not relevant to the node team (we work with git-flow):

```
df = diff
   dc = diff --cached
   lg = log -p
   lol = log --graph --decorate --pretty=oneline --abbrev-commit
   lola = log --graph --decorate --pretty=oneline --abbrev-commit --all
   ls = ls-files
   unstash = stash apply
   stg-clear = push origin :dev_stg_[my name]
   stg-pull = pull --rebase
   stg-push = push origin dev:dev_stg_[my name]
```

to see which aliases you configured, run (after editing the file with that example) **git aliases**.

You can modify that example to push into specific branch or committing to stg (don't forget to change ***[my name]*** !)

Colours!!!

To give your life colour and meaning, add the following to your ~/.gitconfig!

```
[color]
    ui = auto
[color "branch"]
    current = yellow reverse
    local = yellow
    remote = green
[color "diff"]
    meta = yellow bold
    frag = magenta bold
    old = red bold
    new = green bold
[color "status"]
    added = yellow
    changed = green
    untracked = cyan
```

Git Tricks you want to know

Sharing branches other than master

Sharing a private branch with others:

```
> git checkout ira
> git push origin ira
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 579 bytes, done.
Total 5 (delta 3), reused 0 (delta 0)
To ssh://dev-gitsrv/cloudnode
 * [new branch]      ira -> ira
```

Pulling a new remote branch from the "central" repo:

1. do NOT create a local branch of that same name.
2. do a git fetch (or pull) so the local repo knows about the new remote branch.
3. finally, try to check out the branch as if it exists locally. since it does not, git will automagically deduce you are trying to get it linked up to origin:

```
> git checkout ira
Branch ira set up to track remote branch ira from origin.
Switched to a new branch 'ira'
```

Which means your .git/config will now have a section like this:

```
[branch "ira"]
    remote = origin
    merge = refs/heads/ira
```

NOTE! the merge should go to the correct remote reference (in this case refs/heads/ira) and not master or anything unless you really mean it, and you probably don't.

Comitting on a CI machine that has no push permissions

In general CI machines that are not for developpers are supposed to only clone a repo and not push back into the dev-gitsrv or any other "origin". you can still work locally on such a machine, and then pull the changes from your work environment.

Basic functionality for pulling commits on branch "my_feature" only comitted on "my_ci":

```
ssh root@my_dev_machine
my_dev_machine> cd /export/rocks-repo ; git checkout my_feature
my_dev_machine> git pull ssh://root@my_ci/export/rocks-repo my_feature
```

for permanent pulling from a machine other than "origin", just set up a new remote in .git/config like so:

```
[remote "ci"]
    fetch = +refs/heads/*:refs/remotes/origin/*
    url = ssh://my_ci/export/rocks-repo
[branch "my_feature"]
    remote = ci
    merge = refs/heads/my_feature
```

Then you can do:

```
my_dev_machine> git checkout my_feature ; git pull
```

Note! if a branch's default remote is not origin, "git pull" and "push" will go to that remote! you can always force pulling and pushing from a specific remote, e.g.:

```
> git pull origin my_feature
> git pull ci develop
```

Note! Always make sure you are pushing and pulling the right branch when you specify a remote branch. If you chekout my_feature and then pull remote master, it will merge master from the remote machine into my_feature, not into the local master.

Partial commit

Commit specific change chunks instead of all changes in a file:

```
git commit -p
```

Branch + checkout in one go

create a new branch and check it out at the same time:

```
git checkout -b newBranch
```

Hop between branches like a pro

Like "cd -" for changing into the last workdir, "git checkout -" jumps to the last used branch.

More

Look at GIT Usage